FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Gene prediction using Deep Learning

Pedro Vieira Lamares Martins

Mestrado Integrado em Engenharia Informática e Computação

Supervisor: Rui Camacho (FEUP)

Second Supervisor: Nuno Fonseca (EBI-Cambridge, UK)

July 22, 2018

Gene prediction using Deep Learning

Pedro Vieira Lamares Martins

Mestrado Integrado em Engenharia Informática e Computação

Approved in oral examination by the committee:

Chair: Doctor Jorge Alves da Silva
External Examiner: Doctor Carlos Manuel Abreu Gomes Ferreira
Supervisor: Doctor Rui Carlos Camacho de Sousa Ferreira da Silva

July 22, 2018

Abstract

Every living being has in their cells complex molecules called Deoxyribonucleic Acid (or DNA) which are responsible for all their biological features. This DNA molecule is condensed into larger structures called chromosomes, which all together compose the individual's genome. Genes are size varying DNA sequences which contain a code that are often used to synthesize proteins. Proteins are very large molecules which have a multitude of purposes within the individual's body.

Only a very small portion of the DNA has gene sequences. There is no accurate number on the total number of genes that exist in the human genome, but current estimations place that number between 20000 and 25000. Ever since the entire human genome has been sequenced, there has been an effort to consistently try to identify the gene sequences. The number was initially thought to be much higher, but it has since been furthered down following improvements in gene finding techniques. Computational prediction of genes is among these improvements, and is nowadays an area of deep interest in bioinformatics as new tools focused on the theme are developed. Gene prediction is however not an easy task, with many variables conditioning its effectiveness.

Advances in machine learning techniques are expected to improve the prediction and classification of DNA sequences. Deep Learning (DL), in particular, is one of such techniques. DL can be seen as an evolution of the Artificial Neural Networks technology where the training methodology has been improved. It has been shown that DL is quite adequate to handle classification/regression tasks where data sets have a very large number of attributes. That is the case, for example, when large DNA sequences are used as inputs to neural networks.

This project aims towards the study of prediction and classification of genomic sequences using a DL model called multilayer perceptron. Through data mining and machine learning techniques, we trained this model to distinguish genes from other DNA elements. With this work, we hope to promote the use of this type of technologies to create new tools that could handle large amounts of biological data, further improving knowledge in the field of gene prediction.

Resumo

Todos os seres vivos têm na constituição das suas células moléculas complexas chamadas Ácido Desoxirribonucleico (ou ADN) que são responsáveis por todas as suas atividades biológicas. Estas moléculas são condensadas em estruturas maiores chamadas cromossomas, que no seu conjunto formam o chamado genoma do indivíduo. Genes são sequências de ADN com tamanho variado que contêm código que é muitas vezes usado para sintetizar proteínas, moléculas de tamanho considerável que são usadas numa vasta quantidade de funções no organismo.

Apenas uma porção muito pequena do ADN tem sequências que fazem parte dos genes. Não existe um número certo para a quantidade total de genes no genoma humano, mas as estimativas mais recentes colocam esse número entre 20000 e 25000. Desde que o genoma humano foi sequenciado na sua totalidade que existe um esforço para tentar identificar as sequências que constituem os genes. O número de genes, inicialmente pensado como bastante maior, tem levado uma redução ao longo do tempo após inovações em técnicas de deteção de genes. A utilização do computador para previsão de genes está entre essas inovações, sendo hoje uma área de interesse em Bioinformática, com ferramentas focadas no tema a serem constantemente desenvolvidas. A previsão de genes não é, porém, uma tarefa fácil, havendo vários condicionantes à sua efetividade.

É esperado que avanços em técnicas de machine learning possam melhorar a previsão e classificação de sequências de ADN. Deep Learning (DL) é uma dessas técnicas, podendo ser vista como uma evolução das Redes Neuronais Artificiais convencionais, onde o método de treino dessas redes é superior. Foi provado que DL é bastante adequado para lidar com tarefas de classificação/regressão onde os conjuntos de dados possuem um vasto número de atributos, sendo este o caso para os conjuntos de genes que servem como entrada para as redes neuronais.

Este projeto tem como objetivo estudar a previsão e classificação de sequências genómicas utilizando um modelo em DL chamado perceptron em multicamadas. Através do uso de técnicas de extração de dados e aprendizagem computacional, treinámos este modelo para distinguir genes de outros elementos do ADN. Com este trabalho, desejamos promover a utilização deste tipo de tecnologias para criar novas ferramentas que consigam lidar com grandes quantidades de dados de origem biológica, permitindo avanços na área de previsão de genes.

Acknowledgements

I want to take this opportunity to acknowledge a few people which were pivotal not only in the conclusion of this work but also throughout this chapter of my life as a college student which is about to come to an end.

First and foremost, this dissertation couldn't be done without the consistent support from Rui Camacho and Nuno Fonseca, my supervisors. I'm deeply thankful for their mentorship, readily availability and above all the technology freedom they allowed me to have in order to accomplish the desired goals.

Next, I want to acknowledge the whole group of people that were present during my academical life. Some became friends, others did not, but all of them in their own way had an impact in my life and defined me as the person I am today. I won't name all of them because fortunately they are so many and I would risk forgetting someone, but the very few special ones know who they are and can be sure that I could not have reached this far if it weren't for their constant friendship over the years. They put a smile on my face when I needed to and I will always be grateful for that.

Next, my dear parents. For all their unconditional love, financial and emotional support and the occasional well needed slap in the face.

Finally, I don't think that none of this could be possible if it wasn't for the exceptional intellectual ability of one of the most important and decisive but yet sometimes seemingly underappreciated people of our history. The discoveries he made would eventually be put to practice into modern technologies and completely transform the way we see and interact with our world, creating the most important invention of the 20th century. I'm currently writing this document in that invention and I plan on devoting my entire life to it. So for your unprecedented work, thank you very much Mr. Alan Turing.

Pedro Martins

"Success always demands a greater effort."

Winston Churchill

Contents

List of Figures

LIST OF FIGURES

List of Tables

List of Equations

Abbreviations

ANN	Artificial Neural Network
CESAR	Coding Exon Structure Aware Realigner
CNN	Convolutional Neural Network
CRISP-DM	CRoss-Industry Standard Process for Data Mining
DDBJ	DNA Data Bank of Japan
DL	Deep Learning
DL4J	DeepLearning4J
DNA	DeoxyriboNucleic Acid
ENA	European Nucleotide Archive
FFNN	FeedForward Neural Network
FN	False Negatives
FP	False Positives
FTP	File Transfer Protocol
GHMM	Generalized Hidden Markov Model
GVPS	General Vertebrate Parameter Set
HMM	Hidden Markov Model
JDMP	Java Data Mining Package
KDD	Knowledge Discovery from Databases
LINE	Long Interspersed Nuclear Elements
LR	Linear Regression
MLP	MultiLayer Perceptron
mRNA	Messenger RiboNucleic Acid
MSE	Mean Squared Error
NB	Naive-Bayes
NCBI	National Center for Biotechnology Information
NIH	National Institutes of Health
NPV	Negative Predictive Value
PCA	Principal Component Analysis
PPV	Positive Predictive Value
RBM	Restricted Boltzmann Machine
ReLU	Rectified Linear Unit
RNA	RiboNucleic Acid
RNN	Recurrent Neural Network
SDA	Stacked Denoising Autoencoder
SINE	Short Interspersed Nuclear Elements
SVM	Support Vector Machine
TN	True Negatives
TP	True Positives
tRNA	Transfer RiboNucleic Acid

Chapter 1

Introduction

This section begins by presenting the context of this work. Then we introduce the goals and motivations behind the project and also the main approaches that were thought to tackle the problem. We finish this section by explaining what is the remaining structure that this document follows.

1.1 Context

The human genome consists of approximately three billion base pairs and has been sequenced for thirteen years between 1990 and 2003 in a scientific research project called the Human Genome Project [CMP03]. By making the genome available, scientists could discover its constitution and identify which genome sequences were behind the production of proteins. This genome sequences are called genes.

Finding and identifying genes and how they work has been important throughout the years, being fundamental in many medical and scientific breakthroughs. Examples include the discovering of how certain mutations are linked to some forms of cancer, using individual genome information in the application of forensic science, processing of genetically modified food among other utilizations in distinct areas such as agriculture, energy production, archeology, paleontology and biochemical engineering.

Computational gene prediction has been growing in the past two decades and is becoming increasingly important as a method of automatic analysis and annotation of large genome sequences [WZ04], with many software tools developed over the years targeting a variety of organisms. The different treatment of the inputs, the performance when running the application, the accuracy of the prediction, the algorithm used for modeling the gene structure and the way the outputs are shown to the user are all measurements that can be used to compare the existing software tools [Eri15]. Most existing gene prediction software are trained by the developers of these software using gene sets either provided by outside sources or widely available online.

1.2 Goals and Motivations

The main goal of most gene prediction software is to find out whether or not given input (usually sequences of DNA, although other type of biological sequences might be accepted as well) is a protein-coding gene, with most softwares delivering other useful insightful information such as where does that gene starts in the sequence, where does it end and the likelihood of all that data, among other things. Sometimes, this information is inaccurate [MSSR02]. This can be due to a number of reasons, one of them being limitations related with the inherited structure behind the learning process or algorithms used in the creation of some software tools.

Gene prediction can be divided into various domains, each of them having their own foundations and based software tools. Comparative genomics, one of these domains, has the objective of placing genomes of different species side by side to identify potential similarities between them, easing the subsequent gene annotation of the target species. Ab initio on the other hand, is a domain which uses only information retrieved from the sequences alone to identify particular characteristics of that sequence. Gene prediction been extensively studied over the years, with many related software being developed as well for both of the aforementioned domains [SN00].

Finding genes within a genome is not an easy task. It is error prone, with many problems arising from the inconsistency between genes and due to the fact that the genome of most eukaryotic organisms contain a very small amount of genes. Other problems arise from the great distances between gene segments (exons) and the possibility of alternative splicing (different ways of combining those exons).

Deep Learning (DL) is a machine learning subfield which extends the notions of artificial neural networks. Although poorly understood, DL is getting consistently more attention in the artificial intelligence area after being found to outperform the more commonly used classification methods such as decision trees and linear classifiers [Out, LNC+11, Sch15].

The goal of this work is to study the usage of DL techniques on the field of gene prediction and classification. By exploring common DL architectures, we want to create a classifier which can clarify how this type of algorithms behave with large DNA sequences. We hope that this sort of insights can lead to subsequent implementations of new, state of the art gene prediction applications.

Our main intention is to create a model following the usual approaches in machine learning, which include the following steps:

- Find meaningful datasets of DNA sequences which are used in the training process of the model construction. Apart from being ultimately divided into two distinct sets (one used to train the model and the other one used to test it), this data needs to go through some data preparation and pre-processing before being fed to the algorithm.
- The conceptualization and implementation of a DL-based model. This is the longest process, which requires the correct tuning of all its hyperparameters.
- The evaluation of the created model, with a subsequent analysis on the results.

1.3 Document Structure

Besides this introductory chapter, this report has four other chapters which focus on different aspects of the work.

In Chapter 2, we introduce the reader into the biological background concepts necessary to understand the scope of this work. We start by giving a succinct explanation of genomics. Among others, we establish notions such as gene, DNA and chromosome and how they relate with each other. In addition to this, we also provide a brief explanation of gene prediction techniques and a report on a few available dedicated software tools.

Chapter 3 focus on the technological aspects of this project. We throughly explain how data mining and deep learning work, describing the main concepts and topics behind each of those fields. We also provide formulas, images and tables whenever necessary to further improve our explanations. Additional sections present the most commonly used tools for data mining and state of the art deep learning frameworks.

Chapter 4 explains the implementation of the project. Besides describing our research methodology and work environment, we also give details on the creation of the model and the reasoning behind the choice of hyperparameters. We then comprehensively report our experiments and assess and discuss the results.

Finally, Chapter 5 concludes this document by presenting the goal's fulfillment, the most noticeable difficulties as well as prospects for future work.

Chapter 2

Biological Background

This chapter explains the basic biological concepts necessary to understand our project. We start out by giving an explanation on how the cellular biology works and its relation with notions such as genes and chromosomes. Next, we explain gene prediction, its various techniques and different applications in the real world. We end this section by describing some well known software tools used for gene prediction.

2.1 Cell and DNA

Every living being is composed of small biological units called cells. Unicellular organisms consist of only one cell, while multicellular organisms consist of more than one cell. The number of cells is related with the complexity of an organism, where some animals [1] can be comprised of several hundreds of trillions of cells collectively forming higher structures such as tissue and organs [LBZ+95].

Cells are itself very complex, being composed of many parts called organelles. Each organelle has a different purpose, but all together they carry out the most basic biological functions such as processing nutrients into energy. The nucleus is one of these organelles, and serves the cell with instructions which control how it behaves with its environment [Koz83]. Nucleus also contains the organism's hereditary information in large structures called chromosomes. Chromosomes are made of packaged long molecules called DeoxyriboNucleic Acid (DNA) [Cel, Yun76].

DNA carries the genetic information of an individual. DNA is a large molecule composed of nucleotides, where each nucleotide is itself comprised of three subunits: a nitrogenous base, a sugar and at least a phosphate group. The nitrogenous base can be one of possible four - adenine, cytosine, guanine and thymine - and they pair between themselves following some basic rules, creating base pairs. One type of nitrogenous base cannot be paired with another nitrogenous base of the same type, while Cytosine can only be paired with Guanine and Thymine can only be paired with Adenine [PBH+94]. This can be observed in Figure 2.1 [2].

[1] e.g. the blue whale

[2] Image source: https://commons.wikimedia.org

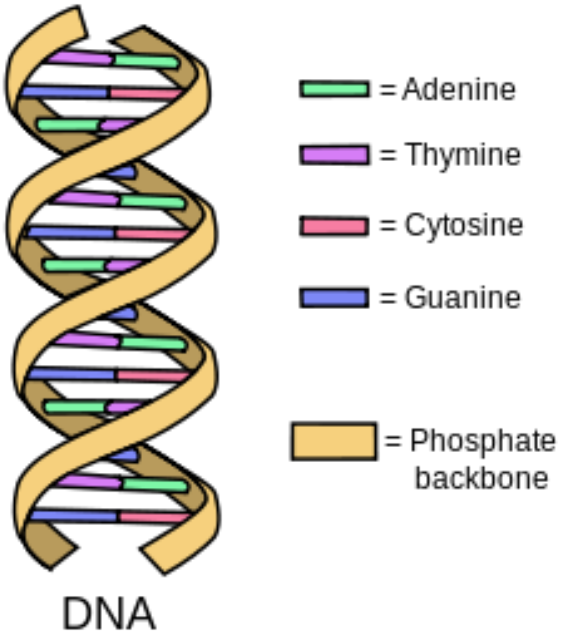

Figure 2.1: DNA structure with the four possible nitrogenous bases

2.2 Genome

The genome of an organism is its genetic material. This genetic material has all the information the organism needs to grow and develop. The size of the genome varies. For example, in the case of humans, there is more than three billion base pairs which condense into all the forty six chromosomes [VAM+01].

The complete list of nucleotides that creates the DNA is called the genome sequence. This sequence is very similar between individuals of the same species, varying only in very small subsets which allows diversity. There is more than 34,000 species [TRCM08] which have their genomes sequenced, the *Homo sapiens* being one of them.

The human genome composition can be seen in Figure 2.2 [3]. The configuration of the genome can be divided into two distinct groups:

- Non-coding sequences, which include elements such as introns, pseudogenes, transposable elements, long interspersed nuclear elements (LINEs), short interspersed nuclear elements (SINEs), regulatory DNA, among others.
- Coding sequences, which carry the necessary information for protein synthesis.

Non-coding sequences account for almost 99% of the entire human genome [BFAW+05]. The remaining sequences - the coding part of the genome - are also called genes.

[3] Image source: https://www.polypompholyx.com/2012/09/a-brief-history-of-rubbish/

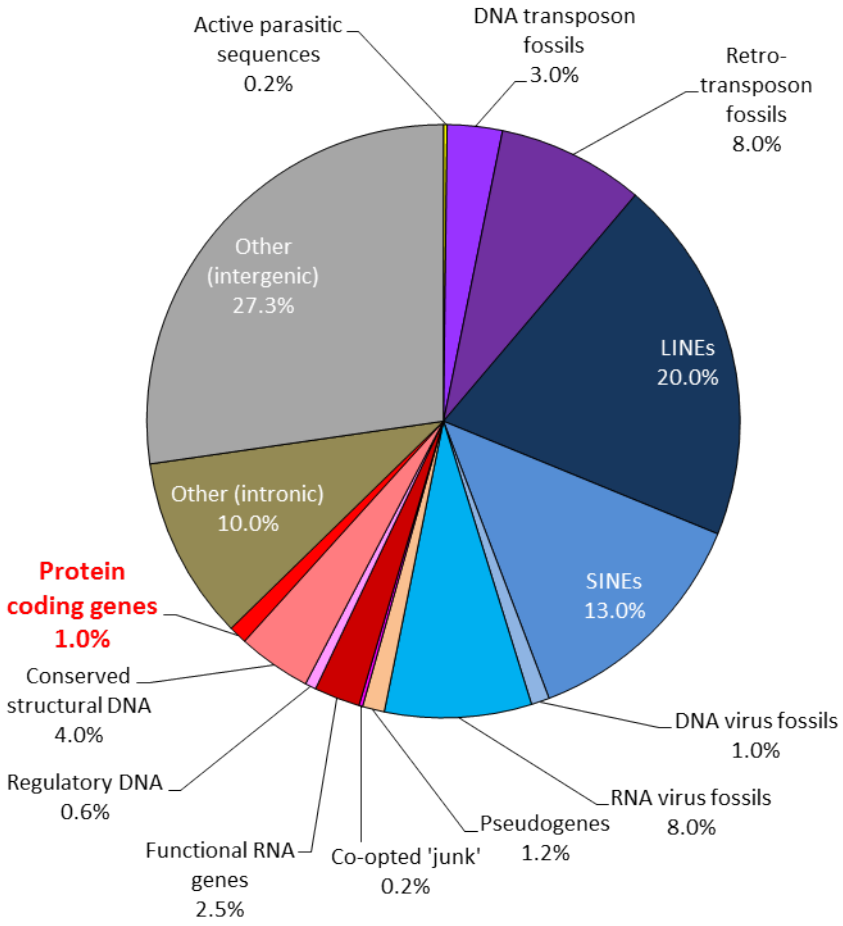

Figure 2.2: Human genome constitution

2.3 Genes and Protein Synthesis

Genes are DNA sequences with sizes varying from a couple of hundred base pairs to more than two million base pairs. Humans are estimated to have somewhere between 20000 and 25000 genes in its genome. These sequences are responsible for the codification of proteins in a process called protein synthesis [VVDVDV+02]. The codification itself is highly similar among all living organisms, being also known as the genetic code. Inside the cell's nucleus, the process of protein synthesis begins by having the DNA helix unzipping into a specific set of instructions. This instructions are interpreted in sequences of three consecutive nucleotides called codons. All 32 possible combinations of codons have a specific meaning that is decoded by the genetic code [AB93].

The synthesis itself happen between two sets of codons: the start codon and the stop codon. At the starting position, the cell proceeds to create a copy of the DNA in the form of a RiboNucleic Acid (RNA) molecule, delivering it to outside the nucleus for decoding. This process is called transcription and the sequence is now called messenger RNA (mRNA). The mRNA sequence can contain coding regions called exons and non-coding regions called introns. Although RNA and

DNA have a very similar structure, RNA is composed of only one string of nucleotides, with Thymine being replaced by another nitrogenous base called Uracil [KFDB01].

The translation is the following step and it occurs in the zone inside the cell surrounding the nucleus called cytoplasm. The introns are removed in a process called splicing, and the resulting sequence is attached by an organelle called ribosome that scans the mRNA to find its start codon. The sequence is read until a stop codon is found. A molecule called transfer RNA (tRNA) serves as the link between the mRNA and the amino acid chain required to produce the proteins. Each tRNA then follows the set of rules present in the genetic code. For example, a sequence of Adenine, Uracil and Guanine (AUG) corresponds to a tRNA carrying the amino acid Methionine (MET) [Cri68]. The process of protein synthesis is illustrated in Figure 2.3 [4].

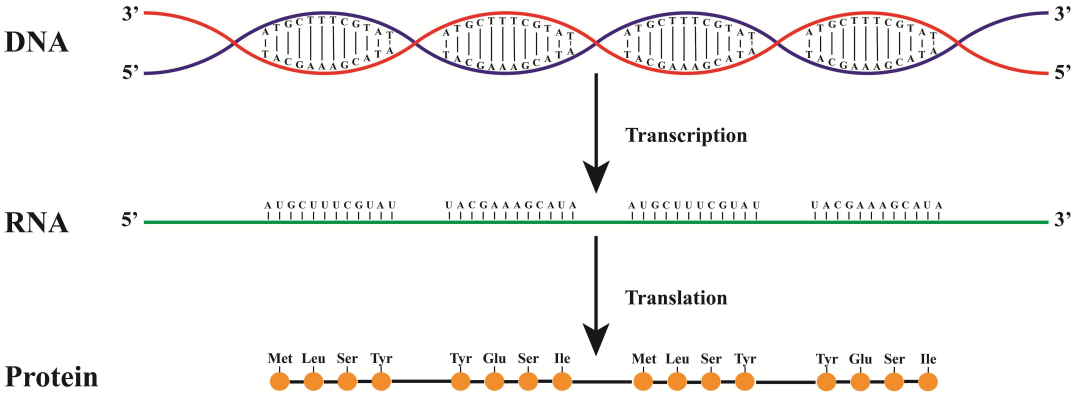

Figure 2.3: Transcription and translation processes of protein synthesis

2.4 Genome Annotation

The concept of genome annotation consists of making an encyclopedic approach to the coding areas of the genome. This involves dividing the genome in portions where the coding and the non-coding areas are identified. These regions are later qualified according to their biological definition [MSSR02].

One annotation is categorized structurally as well as functionally. The structural annotation consists of identifying the physical location of a gene inside the genome as well as its constitution. Concurrently, the functional annotation attempts to describe the gene's biological activity and how it expresses itself to proteins [Sle10].

Gene prediction or gene finding consists of a series of techniques and methods which can be considered as a step in gene annotation. The goal of gene prediction is to find the beginning and the end positions of a gene and other functional regions of the genome [DC06]. This information can then be used to further study the region, eventually leading to its annotation.

[4] Image source: https://www.science-explained.com

Statistical Methods

Statistical or *ab initio* methods attempt to find or predict genes based on the constitution of the sequence. There's statistical significance in some coding regions of a sequence, such as near the starting or ending points of those regions or near transcription regions. The knowledge about these regions can be used to create probability models that can be applied to other similar genes [MD02].

These methods rely on empirical evidence regarding already existing genes. They detect warning spots or signs regarding certain regions of the DNA sequence, which can vary in size. These signs depend on the type of organism. If the organism is a prokaryote [5], the task is usually easy due to the nonexistence of introns in the genome and the appearance of DNA segments which initiate transcription called promoters. In eukaryotes [6], however, the task might prove itself to be harder mostly due to the great distances between exons, the limited knowledge of promoters compared to prokaryotes and the existance of alternative splicing after the transcription [AR02, YKF+09].

Comparative Methods

There are many sequenced genomes which have their genes throughly documented and empirically verified. Comparative or extrinsic methods attempt to discover new genes using the information acquired from those documented genomes. This method derives from the fact that two organisms which aren't taxonomically very far from each other have very similar genes. For instance, although they only share the same class, human and mouse genes can be compared in many ways. Near 95% of both protein coding regions are almost identical, 75% have coding regions with similar lengths and even the non coding areas share around 35% of similarity [NK05].

Most versions of the method begin by aligning the sequences to be compared and try to find similarities between them. These can be either global methods (which compare the sequences over their entire length) or local methods (which search only for certain regions of highly similarity). Almost all of these methods require some sort of cleanup between the comparing sequences to avoid common introns and other non-coding regions inserted between the genes.

2.4.1 Gene Prediction Software

In this section, we will present a few software tools that have been consistently used as reference when discussing computational gene prediction.

GENSCAN

GENSCAN [Kor04] identifies exons and introns in genomic DNA using *ab initio* methods. The algorithms include a homogeneous fifth order Markov model of non coding regions and a three periodic (inhomogeneous) fifth order Markov model of coding regions. Among the existing features, there is the possibility of predicting multiple genes in a sequence, to deal with partial or

[5] Prokaryotes are organisms, mostly unicellular, composed of simple structures without a cell nucleus

[6] Eukaryotes are organisms with more complex organelles bound by membranes, including a nucleus

complete genes, and to predict genes starting on both ends of the DNA input sequences. GENSCAN accepts sequences up to one million characters in length, and accepts parameters from the General Vertebrate Parameter Set (GVPS), among others such as *Arabidopsis thaliana* [7]

GENSCAN can be used via a web user interface [8] and is also available for use in a Desktop. It is not open source, but has been available since 1997 and it is one of the first and most studied gene prediction software which influenced the creation of other existing tools.

HMMgene

HMMgene [HCBS03] focuses on the prediction of human and *Caenorhabditis elegans* [9] genes. It is based on a probabilistic Hidden Markov Model (HMM) model, with its framework allowing a very simple but powerful way of including database matches. The big advantage of this HMM compared with other models is that it recreates the grammatical structure of genes. This ensures starting and ending at the right codons, and prevents alternative splicing among other nuisances.

HMMgene is a web-based tool [10] created in 2000 which is not available for download. The user has many available options to shape the results, including the possibility of downloading the generated outputs. There is no possibility of further model training.

CESAR

Coding exon structure aware realigner (CESAR) [SEH16] is a HMM based software tool created with the purpose of distinguish itself from existing tools by using comparative methods instead of statistical or methods. CESAR is a faster, more memory resourceful tool than most, allowing its use without resorting to heavy computational power. This model enables the ability to find distant exons, increasing the accuracy of its results. It also analyzes and searches full sequences for deleted introns, being able to find entire genes instead of only small and local pieces of them.

Although CESAR can be used for gene prediction, its main focus and the reason that drove into its implementation is a method called pairwise sequence alignment. This method consists in a way of arranging DNA or RNA sequences with the intent of finding similarities between them. It is a recent tool, being available since 2016. It is, however, unavailable for personal use. The accuracy results have been publicized in a paper from Max Planck Institute of Molecular Cell Biology and Genetics.

GeneWise

GeneWise [BCD04] is available as a tool which predicts gene structures using homologous amino acid sequences. It was created with a combination of HMM models and is considered highly accurate, providing many completed gene sequences if served with the right inputs. The tool

[7] A plant native to Eurasia and Africa also known as thale cress
[8] GENSCAN online: http://genes.mit.edu/GENSCAN.html
[9] A terrestrial nematode, the first multicellular organism to have its full genome sequenced
[10] HMMgene online: http://www.cbs.dtu.dk/services/HMMgene/

accepts parameters from the GVPS, plants from the *Brassicales* order and some species of fungi from the *Zygomycetes* class.

This software appeared in 2004 and provided a web user interface [11] with many distinct options from other tools back then. These included the possibility of inputs of two different sequences at the most and outputs about the translation into proteins. The inputs could be sequences o DNA, RNA or amino acids.

AUGUSTUS

AUGUSTUS [SM05] is a tool which predicts genes in eukaryotic organisms with an implementation based on the evaluation of hints to some coding regions of the genome. It is trained by a Generalized Hidden Markov Model (GHMM) that takes intrinsic and extrinsic information into account. This model has been trained by more than seventy organisms ranging from animals to bacteria.

The algorithm ignores conflicts with other existing sequences. It was developed by Mario Stanke in 2003 and is available as a web-based service with a user interface [12] or it can be downloaded to be run locally in Linux systems. Augustus is also open source.

GeneTack

GeneTack [AWJP08] uses a frameshift identification in protein-coding sequences to identify genes in prokaryotic organisms. The program uses the Viterbi algorithm [For73] to find the maximum likelihood path between true genes and sequences which appear to be protein-coding but in fact are not.

This tool was developed in 2009 in the Department of Biomedical Engineering and Division of Computational Science and Engineering at the Georgia Institute of Technology and is freely available for download.

Other tools

Gene prediction software have been appearing ever since the dawn of computing due to ever growing interest in the area. Tools over the years can be divided into four different generations:

- The first generation, designed to identify estimated locations of protein-coding regions in the genome. Examples include TestCode (from 1982) and Grail (from 1992).
- The second generation, which could more accurately predict exon locations by combining splice signal and identification of homologous protein-coding regions. Examples include Sorfind (from 1992) and Xpoung (from 1994).
- The third generation, which predicted complete gene structures instead of localized exons. Some examples include FGENEH (1994) and Genie (1995).

[11] GeneWise online: https://www.ebi.ac.uk/Tools/psa/genewise/
[12] AUGUSTUS online: http://augustus.gobics.de/

- The fourth and current generation, which combined the knowledge obtained from the previous generations with machine learning theory to create faster, more accurate tools. With easier access to computers and with projects being developed more frequently and by smaller teams, there has been an enormous amount of software developed in this generation. AUGUSTUS, GENESCAN, GeneWise, HMMgene, GeneTack and PRODIGAL are just a few examples among several others described at [WZ04] and [PIM+10].

2.5 Chapter Summary

In this chapter, we reported all the biological concepts needed in order to understand the scope of the project. We explained the biology behind genes and the purposes and applications of gene prediction. We also summarized available gene prediction software tools, concluding that the main differences between them reside in the trained models, with some tools being focused on eukaryotic organisms while others are directed towards prokaryotes. The most used and referenced software had a few peculiarities which distinguished themselves from the lesser known tools. This included features such a web-user interface with several available options for the user as well as a higher and more diverse number of organisms that had their genomes used during the tool's training.

Chapter 3

Technological Background

In this chapter, we introduce the technological concepts behind the fields of Data Mining and Deep Learning which were used in the development of our project. We give an overview on both departments: a detailed explanation of all the Data Mining related tasks and algorithms, and an analysis of all the architectures and configuration parameters behind Deep Learning. We also introduce tools and frameworks commonly used in this kind of projects.

3.1 Data Mining

Many state of the art technologies exist today in part due to Machine Learning. Image and voice recognition, personalized marketing, advanced customization and all sorts of data analytics are just a few areas affected by the ability of the computer to learn and understand its surroundings.

Machine Learning is a subfield of Computer Science which has the goal of making the computers act and learn on their own without the need of explicit programming. This learning can only be achieved if the computer works with real data. The field that tries to collect information or patterns from data is known as Knowledge Discovery from Databases (KDD). A KDD process has many related tasks, with one of them being called Data Mining [FPSS96]. Since nowadays the term Data Mining is often used instead of KDD, we will follow this trend and use Data Mining as a synonym for KDD. The usual steps behind Data Mining problems can be seen in Figure 3.1.

The usual steps in Figure 3.1 are just one of several possible approaches to a Data Mining problem. Another common example is a procedure called Cross-Industry Standard Process for Data Mining (CRISP-DM), one of the most widely used methodologies in the industry today with business-oriented tasks such as business understanding and the deployment plan [CCK+00].

Data Mining tasks can further divide the algorithms into two main different groups: predictive learning (where we look at the current data and try to understand the future) and descriptive learning (where we analyze the current data and try to give insights to its meaning). Some sources also identify a third group called prescriptive learning, which consists of looking at the data and trying to obtain answers and advices on possible outcomes to decide what to do next [Lej01].

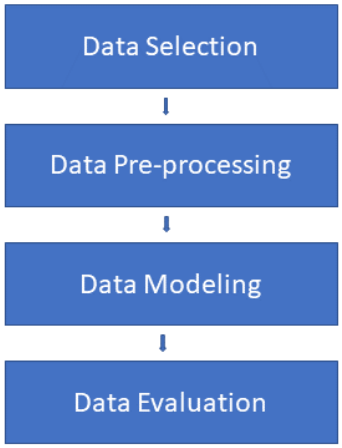

Figure 3.1: Usual sequence of steps in a Data Mining problem

Supervised and Unsupervised Learning

Supervised Learning consists of the task of inducing a function from labeled training data. It is the most commonly used machine learning approach. In supervising learning, you have input variables and an output variable and the goal is to use the algorithms to learn how to map the inputs to the outputs.

Unsupervised learning, on the other hand, differs from supervised approaches because the output variable is unknown or not given. The goal is to structurally model the data in order to understand more about it. The name derives from the fact that there is not a correct answer in the end, and the algorithms have to decide for themselves what to do with the analyzed data and attempt to discover any meaningful connection between all the records [Don11].

Some sources identify semi-supervised learning as another learning type, which sits between supervised and unsupervised learning. It happens when you have a large amount of input data yet only some of the data are labeled, the goal being to try to learn from both types of input [CSZ09].

Data Mining Tasks

In this section, we will look into some universally used tasks and terms in Data Mining related with descriptive and predictive learning [Han07].

Classification

The goal of classification algorithms is to assign classes or labels to records which haven't been seen before (called test set). These type of algorithms work with a given known collection of records called training set and they work through a model or function which learns on how to map object values to the labeled attributes. This is a type of predictive task.

Regression

Another type of predictive task, prediction algorithms work in a similar manner with classification algorithms, but while classification mostly works with discrete data and maps their results into categorical class labels, prediction models return possible outcomes by handling continuous values.

Time-Series Analysis

This type of predictive task works with time as a variable to try to understand time-series data, which are collections of data sequences over a time period without a consistent meaning. It is the algorithm's purpose to try to unravel that meaning [Ham94]

Clustering

This task works on collections of data, mostly unlabeled, to try to identify similarities between the records and subsequently bundle the data together in groups called clusters. This association is followed by an attempt to attribute a meaning or understanding to those groups through conceptual clustering. This is a type of descriptive task.

Summarization

This sort of descriptive task has the goal of depicting data in a more compact and concise way comparing to its raw representation without losing any of its original information. This can result in the creation of alternative, simplified data presentations which can lead to other insights not previously noticed.

Association Rules Discovery

The discovery of association rules can be described as a descriptive task as well. The purpose is to find associations and rules between the elements of a set of records. This elements need to be related between themselves, and that relation must be contextualized [AS+94].

Sequential Patterns Discovery

Working on a similar fashion as Association Rules, patterns discovery in sequential data also uses time as a constraint to infer dependency rules among related records.

3.1.1 Data Selection

Data selection is the paramount phase in Data Mining. One can only hope to achieve state of the art results if a suitable and appropriate dataset exists.

The whole purpose of data selection is to collect the data that is used as the dataset in the remaining process. The usual flow of actions can be seen in Figure 3.2. Depending on the problem

itself, we can have data of different types and representations, with images, video, audio, text or sequence being the most commonly found types. The amount of data required can vary and is usually determined by the mining algorithm to be used as well as the goals behind the project. Deep learning implementations, for instance, are a very greedy type of algorithms that work best when fed with large amounts of data. On the other hand, low complexity models with fewer parameters such as Linear Regression (LR) or Naive-Bayes (NB) can work well with small amounts of data.

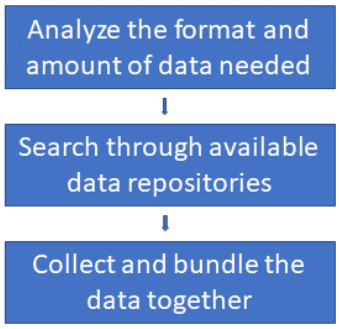

Figure 3.2: Sequence of steps when selecting appropriate datasets

Available data repositories

Relevant amounts of data must be found in order to evaluate our solution later on. Although in some cases data can come from private sources or are collected purposefully by other external entities working for the project, there is also the possibility of searching and aggregating data from public, free, widely available repositories.

When looking for genomics related data, there are many available options. In the context of this project, we will be looking into samples retrieved from sequence databases that provide annotated genomic sequences of a vast collection of genomes. We give details about two universally used repositories in the field of genomics as well as two other examples of smaller and more focused sources.

GenBank

GenBank is a free, public collection of all available DNA sequences. It is served by an online platform [1] that provides easy access to every existing sequence on its database through a powerful search mechanism. At the time of this writing, GenBank had more than 200 million entries in the form of DNA sequences which amounted to almost 263 billion base pairs [GEN]. It is maintained by the National Center for Biotechnology Information (NBCI), a United States branch of the National Institutes of Health (NIH), together with synchronized contributions from Japan's DNA Data Bank of Japan (DBBJ) and Europe's European Nucleotide Archive (ENA). Among the existing features, GenBank supplies the DNA's correspondent protein translation sequences and gives the user the possibility of downloading large sets of records at once [BKML+08].

[1] https://www.ncbi.nlm.nih.gov/genbank/

Ensembl

The Ensembl project dates back from 1999, a few years before the human genome was fully sequenced. It was created with the intention of allowing a comprehensive annotation of the genome once its sequencing was completed, and eventually received an online platform [2] where the users could submit their contributions or search for information about a particular position in the genome. The project evolved over the years and today, maintained by almost 50 people, provides a graphical interface where the users can browse the entire genome of multiple organisms. The available options range from homologue genes searching to the possibility of downloading text-based representations of DNA or amino acid sequences (among others) from a File Transfer Protocol (FTP) service [HBB+02].

GENCODE / NONCODE

GENCODE and NONCODE, although not related, are two projects focused on supplying very specific types of datasets to public availability. Both projects also directly provide their results to other more widely used databases, such as Ensembl and GenBank. The GENCODE project [3] provides gene sequences and annotations retrieved from human and mouse genomes. It contains many files in common formats, ranging from protein-coding sequences to long non-coding transcript sequences [DJB+12]. The NONCODE project [4], on the other hand, provides a very rich collection of datasets from 17 species, consisting of only non-coding samples [BYS+11].

Biological sequences formats

When working with DNA sequences and its related concepts (gene expression, genome annotation, protein synthesis), there is a few default formats that most data appear in. Some of these formats have a piece of text identifying the purpose of the sequence and all of them are in ASCII text. The ones considered the most relevant for this project are reported in the next subsections.

FASTA / FASTQ

One file with the FASTA format represents a nucleotide or a peptide [5] sequence. It is a very simple and trivial format, with only one line describing the sequence and the rest consisting of the sequence itself. An example can be seen in Figure 3.3. It was developed in 1985 when the field of bioinformatics was still starting with the intention of providing a standardized format which would be portable and easy to parse. Online data repositories such as GeneBank and Ensembl provide raw entries in the FASTA format. Besides '*.fasta*', FASTA files may come with the file extension '*.fa*', '*.fsa*', '*.ffn*', among others. Usually each of those file extensions has another meaning to it (e.g., an '*.ffn*' file means the described sequence is that of a coding region in the genome).

[2] https://www.ensembl.org
[3] https://www.gencodegenes.org/
[4] http://www.noncode.org/
[5] Smaller molecules compared with proteins, but not as small as amino acids.

```
>NR_109833.1 Homo sapiens prostate cancer associated non-coding RNA 1
coding RNA
AAATCTCAGCCTCCCACTCCCATATTTACAGTTTGATTAGGGAGGCACATTTAGATATGCAGTGATAATT
GCTTTGCTTAGAGAAATTCAAGGTGATAGAATGCATGGTGACACCTAACCCAGACTGGTAGAGAAAGGGA
ATTCTTCCACTGGGAATGACATGATTATCTAAATAAGTAGGCTCAATCAGGTCAGGAAAGGGCCTGAAAG
ACTATTTCAAGCAGAGGGAAGGTATTTGCCAAGGCCAGGGTGTGTAGTGGAGAGAATGGGCAGTGGCAGA
GAATTATGAAGTGTTCCAATGACTAAAAGTAAAGTAACAAGTTCCTTGTCCAAGAGCTTGGATTGTATCC
TAACTGAAATGAGTATACACTAAGTGTTTGAAGAAGAGGGATGAAATGGTCAAGTTTTCATTACACAAAA
ATAACCTGTTCCTTTCATTTTATGTTTATTTATTTTTTTAATTTTCTGACTGCTCCTTTCTGGAAATCTC
AAATTTATATTTGCCAAATATTGTCACATTTTCGATGGAGAATACAAACTAAGAATGGGTTAGGGAACTG
AGTCAGAAAGTCCCTGTTGTACAATTCAATCATGTTTTTCTAAGGATGTGCTTTTGGACATTATGGAAAC
TATCTTAGGCTCTCACTTGGATCCTAGAAAAGAAGGCACCTGTTAAAAGGAATGTCCAGCCCCACCTAAT
TTGGCAGCTGCCCCTCCAAGCTAATGACATTAGGGATGTAGTGGATTCAAGAGGCTGATGATGCCATCTG
```

Figure 3.3: A partial sequence in a FASTA format retrieved from GeneBank

FASTQ is an alternative version of the FASTA format that has a few changes and adds additional information, but keeps the simplicity and easiness to parse of the original [DG11]. A file in FASTQ has the following configuration:

- The first line is similar to the FASTA format but starts with an '@' instead of a '>'.
- The second line is the sequence itself.
- The third line contains an optional description preceded by a '+'
- The fourth line is the same size as the second, but trades each nucleotide character for a special encoding called "quality values". This encoding ensures the sequence is recognized by software such as the Illumina Genome Analyzer [CFG+09].

Both FASTA and FASTQ (and other formats as well) follow a simple encoding with its characters, with each having a meaning depending on the context of the sequence (DNA, RNA or amino acid results after translation). The significance of all the characters can be checked in Table 3.1. FASTA is the most widely used format in the field of gene prediction and most state of the art software tools accept files in that format.

GCG

The GCG format is used as an alternative in some software, but the premises are the same. The configuration inside a GCG format file has a set of different rules, such as space for comments when separated by two dots and meta-information containing the sequence length, name, date and type (nucleic or amino acid). GCG files, unlike the FASTA format, have only one general suffix , '*.gcg*' [SWT+98].

GenBank format

GenBank, besides accepting and providing raw sequences in the FASTA format, has also a special format created for its databases with the same name as the platform, GenBank. It does not differ

much from both FASTA or GCG, but configures the file a little further by inserting strings at the beginning and at the end of the sequence and throughout the rest of the file, with descriptive names such as "*ORIGIN*" and "*DEFINITION*" [SAM].

Table 3.1: FASTA / FASTQ characters significance with nucleotide and protein sequences

Character in Fasta	Translation (nucleotide)	Translation (amino acid)
A	A (Adenine)	Alanine
B	not A (either C, G, T or U)	Asparagine
C	C (Cytosine)	Cysteine
D	not C (either A, G, T or U)	Aspartic acid
E	-	Glutamic acid
F	-	Phenylalanine
G	G (Guanine)	Glycine
H	not G (either A, C, T or U)	Histidine
I	-	Isoleucine
J	-	Leucine
K	G, T or U (ketone bases)	Lysine
L	-	Leucine
M	A or C (amino groups)	Methionine
N	A, C, G, T or U	Asparagine
O	-	Pyrrolysine
P	-	Proline
Q	-	Glutamine
R	A or G (purine bases)	Arginine
S	C or G (strong bases)	Serine
T	T (Thymine)	Threonine
U	U (Uracil)	Selenocysteine
V	not T, not U (either A, C or G)	Valine
W	A, T or U (weak bases)	Tryptophan
Y	C, T or U (pydimidine bases)	Tyrosine
Z	-	Glutamine

Plain format

Sequences may also come in raw format without any other attached meaning. The only rule they have to follow in order to be correctly parsed by common gene prediction software is to have no spaces whatsoever between the characters. The sequences must not have other characters besides the Latin alphabet as well. Numbers are not allowed, and although lower case characters are accepted, standards from other formats recommend upper case letters.

3.1.2 Data Pre-processing

Another important step in Data Mining is the pre-processing of the dataset once it is collected and properly assembled. Since deep learning algorithms need sizable amounts of data to work as expected, most datasets in this kind of environment have to be huge. The larger a dataset is, the more pre-processing it needs. We cover a few of the topics that usually come with this much data below, where we also present their associated problems and explore commonly used solutions. As can be seen in Figure 3.4, although most steps in data mining are sequential, data pre-processing does not follow a general order, with the only conditions being a raw dataset at the beginning and a prepared training and testing set at the end.

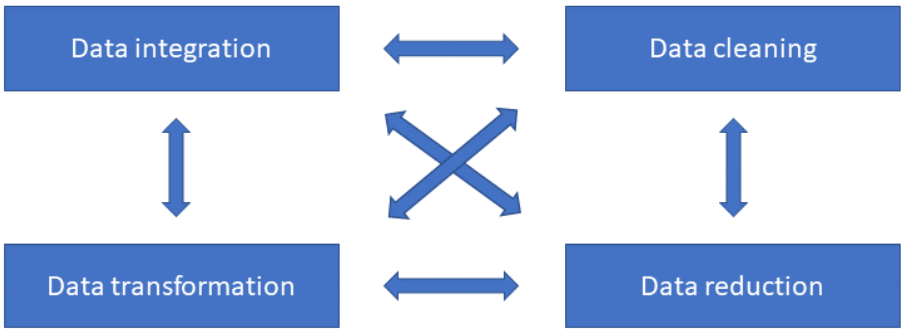

Figure 3.4: Common steps in data pre-processing.

Data integration

The process of data integration consists of approaching data from several different sources and re-arrange it in a way that consistency and coherence can be achieved [Len02]. In the context of this project, it is important that data with different formats (for example, a FASTA file and a GCG file) can be equally put together in the final training dataset before constructing the model. This can happen by converting one format to the other or, uncommonly, by creating an entire new one that joins information from both. In the case of conversion, it is usually made from the more informative or complex format to the simpler one in order to avoid missing information.

Data cleaning

An important process regarding large sets of data, in particular those obtained from many sources (as is the case of sequences archived at repositories such as GenBank) and those which emulate real-world data, is to address faulty entries from these sets. Faultiness may arise from incomplete (cases where the affected samples lack some attributes of interest), inconsistent (cases with attribute values that are different from the expected, e.g a float instead of a string or a negative value when the constraints specify it must be positive) or noisy records (cases where the values are just wrong, or differ much from the rest).

The goal of data cleaning is to identify and either correct, replace or simply remove records which deviate from the expected. There are a few local methods or procedures dedicated to a particular issue in the process of data cleaning. Noisy data for instance, can be identified and promptly removed by comparing each record with its neighbors with the help of algorithms called binning methods [MMG12]. Uncompleted instances can be fixed by several approaches, either by disregarding them or by filling the missing values with content that would not affect the subsequent model [HPK11].

Data transformation

Transforming the data consists in a series of actions that have the goal of standardizing the values from the samples before feeding them to the model. There are many ways to transform data. We look into two of them, data normalization and data generalization.

Data normalization comprises the establishment of upper or lower bounds to the values and ensure the dataset follows those restrictions.

Data generalization is the process of removing specificity, by turning more specific and local data to their correspondent general cases.

Data reduction

Mining high-dimensional data, such as is the case in the field of genomics, can be computationally expensive. When dealing with this type of data, in order to potentially increase the performance of the models that will work with it, is important to find ways to create a more compact data representation compared with raw reads. Data reduction techniques deal with that challenge by analyzing and conceiving ways to condense the data without compromising the integrity of the actual reads. We present common practices which seek to either reduce the size or the number of attributes in the data.

Dimensionality reduction by data removal

In this case, we try to reduce the volume of the data by eliminating entries in the dataset which prove themselves to be redundant, irrelevant or peripheral comparing to the rest. This differs from the process of noisy data cleaning because we are not looking for faulty samples but samples which, although with the correct specifications, don't bring anything new to the mining process.

Numerosity reduction

This technique implies the total replacement of the dataset for a smaller numeric representation. This can be through parametric or nonparametric methods. The former ensures the storage is made through a model estimation such as regression and log-linear models, storing only vital

data attributes information instead of the actual data, while the latter uses concepts such as data sampling and histograms to approximate sparse, sizable data [XKS+06].

Data compression

This type of methods have the intention of reducing the dataset size through encoding mechanisms. One algorithm created for this purpose is called Principal Component Analysis (PCA). This algorithm is computationally inexpensive and through a set of linear calculations tries to find lower dimensional vectors that can be interpreted as the raw data. By being cast into a smaller representation, the data is dimensionality reduced, which not only gives shorter training durations in the mining process but may also provide insights and interpretations for further studying not contemplated before [AW10]. The usage of PCA is available in many state of the art data mining tools. An alternative version called KPCA is also known to provide great results.

3.1.3 Data Evaluation

Evaluating a model once it has been trained is one of the final steps in any data mining process. There are some available options to do this. A confusion matrix, a typical performance measurement, can be seen in Figure 3.5.

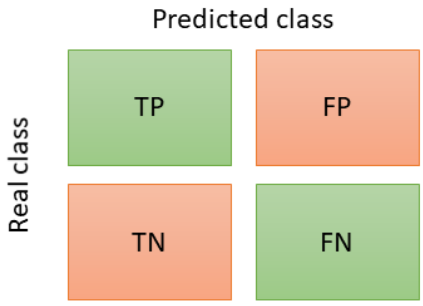

Figure 3.5: A confusion matrix in a classifier with two classes

Confusion matrix

A confusion matrix, also known as error matrix, displays the number or fraction of correct and incorrect predictions made by the model after being trained, comparing the results against the actual known classifications of the testing set.

True Positives

True Positives (TP) is the number of positive cases correctly predicted by the classifier.

True Negatives

True Negatives (TN) is the number of negative cases correctly predicted by the classifier.

False Positives

False Positives (FP) is the number of negative cases incorrectly predicted by the classifier.

False Negatives

False Negatives (FN) is the number of positive cases incorrectly predicted by the classifier.

Classification metrics

There are classification metrics that use the aforementioned confusion matrix values to calculate the performance of the classifier. These are extensively supported in most data mining tools and DL frameworks. We present the most commonly used below.

Positive Predictive Value or Precision

The Positive Predictive Value (PPV) is a metric which gives the fraction corresponding to the positive cases correctly predicted by the classifier (Equation 3.1). This value is also known as precision.

$$PPV = \frac{TP}{TP+FP} \tag{3.1}$$

Negative Predictive Value

The Negative Predictive Value (NPV) is a metric which gives the fraction corresponding to the negative cases correctly predicted by the classifier (Equation 3.2).

$$NPV = \frac{TN}{TN+FN} \tag{3.2}$$

Recall or Sensitivity

The recall or sensitivity is a measurement which gives the classifier's successfulness in predicting positive cases (Equation 3.3).

$$Recall = \frac{TP}{TP+FN} \tag{3.3}$$

Accuracy

A general metric which uses every value in the confusion matrix to show how effective a classifier is (Equation 3.4).

$$Accuracy = \frac{TP + TN}{TP + FP + TN + FN} \tag{3.4}$$

F1 Score

Uses both recall and PPV to calculate their harmonic mean (Equation 3.5). It is usually needed when we are trying to balance between the aforementioned values or when there is class imbalance with a large number of negative entries [BJCF07].

$$F1 = 2 \times \frac{PPV \times Recall}{PPV + Recall} = \frac{2 \times TP}{2 \times TP + FP + FN} \tag{3.5}$$

3.1.4 Classification Algorithms

In the context of gene prediction, we are always dealing with classification problems. When presented with a single sample, the goal of the computer is to decide whether or not that sample falls into the gene category. This is done through machine learning by feeding dedicated models or algorithms with sufficient amounts of data. We present in this section a few of those algorithms, in what they consist and their main advantages over the others.

Decision trees

This method, which creates a graph-like structure that looks similar to a tree, can be implemented by algorithms such as ID3 or its extension, C4.5. In a common decision tree, we can have three types of nodes: decision nodes, chance nodes and end nodes, which are always at the end of the tree (leaves). Flowing between nodes is ensured by decision rules. When an input reaches an end node, it is classified with the value associated with that node. Besides classification trees, there are other types of different trees and implementations, such as regression trees, boosted trees, rotation forests, among others [RKA06]. An example can be seen in Figure 3.6 [6].

Decision trees hold many advantages over other classification algorithms. Besides being very easy to understand, they work well with small datasets which usually require little pre-processing. They also perform well with larger amounts of data and are very maintainable, with new options and attributes being very easy to add in subsequent updates. However, they can rapidly become complex, arising the need of cutting out branches through pruning or other methods. Another

[6] Image source: https://www.lucidchart.com/pages/decision-tree

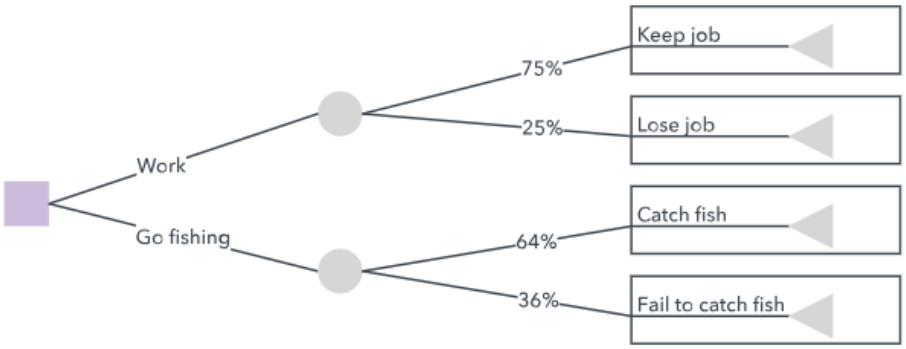

Figure 3.6: A typical decision tree. This is a type of classification tree

disadvantage is its lack of robustness, with the final result being sensitive to small changes in the training set [RM08].

Bayesian networks

This type of networks are based on Bayes' Theorem, which is shown in Equation 3.6. This theorem describes the probability of an event happening having knowledge of something related with the event that has happened before. This prior knowledge is also known as facts. Bayesian networks work around this theorem, being considered an automatic application of the theorem to complex systems. Through the implementation and optimization of its structure, this type of algorithm handle concepts such as the Local Markov property, the Markov blanket or causal networks [ATS03]. Although easy to implement, full machine learning through Bayesian networks can be quite computationally expensive, particularly with high-dimensional data as it happens in the genomics field. The theory behind it also involves many mathematical formulas which may be hard to grasp at first.

There is another learning algorithm based on Bayes' Theorem called NB classifier. They work around feature's conditional independence between each other to create a classifier that connects the samples with a set of labels. While all dependencies in a Bayesian network have to be considered and modeled, the NB model states that all variables are completely independent [M+06].

$$P(A \mid B) = \frac{P(B \mid A) \times P(A)}{P(B)} \tag{3.6}$$

Support Vector Machines

A Support Vector Machine (SVM) plots all the samples into a n-dimensional space (with 'n' being the number of existing labels or features which we are considering) and assigns each sample's value to a coordinate in that space. The algorithm then tries to find the line, also called hyperplane, that distinctively separate all the existing classes from each other. SVM is very efficient

as a machine learning algorithm, and is used for a number of applications ranging from speech recognition to faulty card detection [KAH+14].

Instance-Based

Instance-Based classifiers are a particular type of machine learning method because they do not learn from a trained model. There is no structure whatsoever. Instead, they store their knowledge in the form of instances or memories when doing the training process, and these are subsequently used to classify new incoming instances. It is also known as lazy training [LDR00]. It is a very costly algorithm for large datasets, but usually converges fast with smaller ones. Example methods which use Instance-Based concepts include the k-nearest neighbor and the locally weighted linear regression [ZZ05].

Artificial Neural Networks

Classifiers based on Artificial Neural Networks (ANNs) are very complex and can be further divided into a lot of different yet related concepts. They are loosely based on the brain's neural network. These type of algorithms are behind most state of the art applications in artificial intelligence and are usually chosen when dealing with very large datasets. They are also the main architecture behind deep learning techniques and are further explained in subsection 3.2.1.

3.1.5 Data Mining Platforms

There's several existing Data Mining tools, with some of them being free to use while others requiring a paying license. We list some of them here, particularly those who offer exclusive features or are widely used by the community.

RapidMiner

RapidMiner Studio [AH12] is an open-source desktop software first developed in 2006 which supplies a wide array of data mining options to the user. Besides allowing every data mining step from data access to data evaluation, the software also provides the following main features:

- Very easy to use with a simple, intuitive user-interface.
- Rich collection of data connectors which can work with more than sixty different file types.
- Load and information extraction from unstructured data.
- Graphical options for data exploration.
- Integration with R and Python.
- Extensible with hundreds of available extensions.

The software is free for non-commercial purposes, and the source code is available through a general public license on GitHub. The free version establishes a limit of 10.000 rows per project, by which time the user must decide to obtain the property editions in order to continue. It is one of the most used data mining tools by the time this was written, especially in academical and small-scale projects. An extensive documentation is available in the main website [7] and many other tutorials can be found elsewhere on the Internet.

WEKA

Waikato Environment for Knowledge Analysis or WEKA [HFH+09] is one of the most popular data mining tools available alongside Rapidminer, and is completely free to use under a general public license as well. Unlike Rapidminer however, WEKA may be used without any drawbacks. It was developed in a New Zealand University back in 1993 and was originally written in C, but has since been remade from scratch using the Java language. It has all the functionalities of RapidMiner, but lacks a few specific features such as sequence modeling, automatic parameter optimization in machine learning or model validation using independent testing sets. It is very fast however, can connect with any database through the Java database connectivity, and being written in that language allows for portability throughout other operating systems as well.

R (Programming Language)

Unlike WEKA and Rapidminer, R [RPR] is an interpreted standalone programming language primarily used for its data mining and statistics capabilities. It is an implementation of the S language and provides some powerful features such as embedded operators for calculations in matrices, vectors, arrays and other structures. It is also multi-paradigm, supplying all the usual concepts of a programming language such as functions, exceptions and control flow. R is open-source, compatible with many operating systems and easily integrated with other environments. Several companies use R for their data analysis, examples including Facebook, Google and Twitter.

Other options

KMINE is yet another free option with many graphical features that is also widely used. Besides the aforementioned tools, there are others available, some of them being purpose-specific. OpenNN [Lop12] is a library written in C++ which implements neural networks concepts applied to data mining processes. GATE [GCW+96] is a suite implemented in Java dedicated to natural language and information extraction. Chemicalize [Vie13] does text mining and predictions on chemical notions. There is also other property licensed software provided by big companies such as Microsoft, Hewlett-Packard and Oracle.

[7] RapidMiner documentation: https://docs.rapidminer.com/

3.2 Deep Learning

Deep Learning (DL) can be briefly explained as a machine learning subfield that works with algorithms which structurally and functionally resemble a brain. DL is a very broad subject, having many distinct concepts and notions that need to be understood before being put to practice. In the next sections, we will explain and give examples of some of these concepts and how they relate with one another.

3.2.1 Artificial Neural Networks

The human brain is formed by a collection of billions of small nerve cells called neurons. These cells are complexly connected to countless others by structures called axons. Each neuron accepts or rejects external stimuli by small components of the cell called dendrites. Once an electrical impulse is detected, it quickly travels through the network, with each neuron's axon sending the signal to the connected neuron's dendrites in a process called synapse. Artificial Neural Networks (ANN), like the brain's neural network, have specific structures connected between each other.

In ANNs, the nodes can be seen as the neurons. Each neuron connects and interacts with at least one another through links. The inputs are processed and passed along through the network, eventually reaching a final state where all the computed values are shown to the user [Yeg09]. A typical ANN can be seen on Figure 3.7 [8].

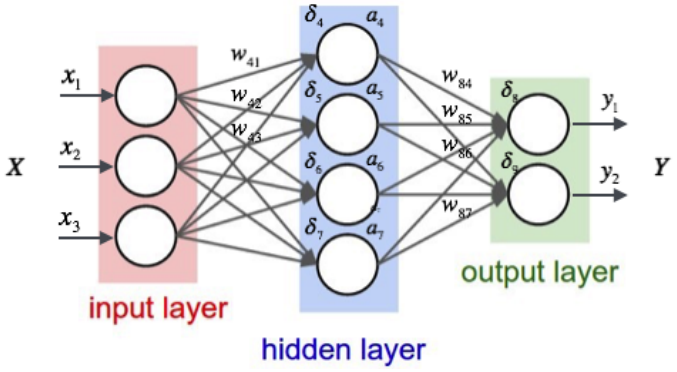

Figure 3.7: An artificial neural network

On a higher level, a simple ANN has at least three distinct layers. The input layer is unique and consists on the collection of nodes to which we provide the input values. The hidden layer has all the intermediate nodes where all the computation occurs. Finally, an output layer, also unique, reveals the calculated results. There can be more than one hidden layer. A number superior than two hidden layers is usually the definition provided for DL architectures and what distinguishes them from other ANNs. The level of abstraction that is achieved by having calculations made in more than one hidden layer is pointed out as the main reason behind DL's state of the art results in many different fields [DVB93].

[8] Image source: https://medium.com/@curiousily/tensorflow-for-hackers-part-iv-neural-network-from-scratch-1a4f504dfa8

Figure 3.7 presents a common type of ANN called FeedForward Neural Network (FFNN), where all the information flows unidirectionally, without any loops. The contrary are called Feed-Back Neural Networks. FFNNs have the following features [BG94]:

- All the nodes in the input layer act as inputs for the nodes in the hidden layer. If there is a second hidden layer, that layer receives the inputs from the first layer and so on. The last hidden layer provides inputs for the output layer. A specific type of FFNN that has more than one hidden layer is a deep architecture called Multilayer Perceptron (MLP), which is further described in section 3.2.5.

- Excluding the output layer, every node in a particular layer is connected to all the nodes in the subsequent layer.

- One connection between two nodes is characterized by a value called weight.

Figure 3.8 [9] is a closer look to how a node behaves. Since this nodes resemble neurons in many ways, they are also commonly called artificial neurons.

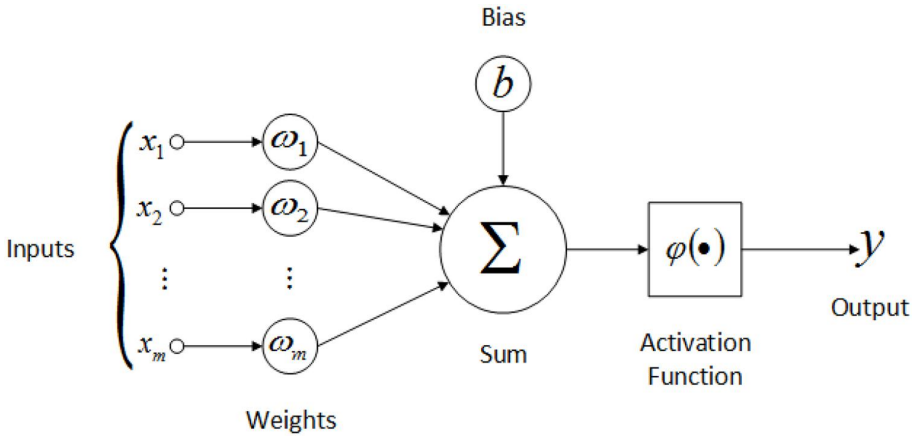

Figure 3.8: Behavior inside each artificial neuron

Each artificial neuron receives a set of inputs, with each of this inputs being subsequently multiplied by its corresponding weight. After all the inputs are processed, the values are combined together and added resulting in an intermediate value called sum or net. The sum is then passed through an activation function which produces the final output of that single node. This output serves as input for one or more connected nodes in the next layer, where they will use that value in a similar manner for its inner calculations.

In every machine learning project, one of the most important phases is the training process. This training ultimately determines the utility of a model and how it works when properly tested. There are, however, frequent known issues that can arise when training a model from scratch, some of which we describe below.

[9] Image source: http://goo.gl/M2BTE4

Overfitting

Overfitting is one of the most common problems in machine learning and consists in fitting the data too closely or narrowly, preventing the model from generalizing or creating new potential hypothesis from samples yet to analyze. Overfitting usually happens with very complex models with a large numbers of parameters [Haw04].

Underfitting

Underfitting, the contrary of overfitting, may also occur within machine learning problems. In this type of cases, the model which is either too simple or was trained with sparse or uninformative data, can't depict a relation between the dataset's features and the target value, resulting in overgeneralizing.

Vanishing gradients

Particularly in a DL model, there may be several hidden layers involved. In gradient-based backpropagation training methods which are further explained in subsection 3.2.2, one problem that can occur is the vanishing of the gradients as we pass information along the layers, eventually reaching a point of which it may become so small that weights no longer are affected by them. Once this happens, the model can't further learn. Some proposed alternatives to the common backpropagation methods claim to solve this problem, as can be seen in [Mar10].

3.2.2 Backpropagation

Training an ANN is usually done with a method called backpropagation. The goal of this technique is to optimize the weights of the nodes in the network by calculating a concept called gradient. The algorithm can be described by the following steps:

1. Weight initialization at the input layer.
2. Forward propagation of the weights in the network with each node using its inputs and associated weights to calculate the activation values.
3. Calculation of the loss function at the output layer.
4. Backward propagation, where the gradients of the loss function are calculated and each layer specific parameters are updated.
5. Repeat the steps 2, 3 and 4 until the stop criteria is met, usually until the loss function is minimized without achieving overfitting.

Some of the aforementioned steps has some keywords, such as weight initialization, activation values and loss function, which are further explained in subsections 3.2.3 and 3.2.4.

Alongside backpropagation, an optimization algorithm for finding local minimums in functions called gradient or steepest descent is popularly used in the training process. There are three types of gradient descent commonly seen in machine learning and DL projects: batch gradient descent, stochastic gradient descent and mini-batch gradient descent. These algorithms can be optimized by methods such as Nesterov momentum updater, ADAM, AdaGrad or RmsProp. They are differentiated by the way they change how the equations associated with gradient descent work.

3.2.3 Hyperparameters

The training process may prove to be quite difficult to do due to a high number of initial variables called hyperparameters. These variables determine the network both structurally as well as functionally, and can range from the activation functions inside each node to the learning rate of the whole algorithm. These are defined at the beginning of the process before training and are unique to a certain problem, depending on factors such as the chosen architecture and the size and format of the dataset. We analyze the main hyperparameters defined for a typical DL model in the next sections.

Network size

The number of hidden layers and the number of nodes per each layer (including the input and the output layers). A small number of hidden layers may result in underfitting, while many hidden layers can have positive outcomes at the expense of more computational time needed. The number of nodes at the input layer is dependent on the amount of data and how we want to model it, while the number of nodes at the output layer is determined by how we want to visualize the processed information.

Weight's initialization

As information is computed through the network, the values are passed from node to node with each weight being equalized in the process. In the beginning however, we need to initialize the network's layers to values different than zero. There are many available options to do this, from completely random initialization to uniform distributions. A common algorithm is called Xavier, which determines the scale of initialization built around the number of nodes in the input and the output layers [GB10].

Dropout

The percentage value of hidden layer nodes which are ignored by the model is called dropout. This is a well known method to avoid overfitting. The value must be properly set, since small percentages may have no effect while bigger ones may result in underfitting instead.

Learning Rate

This value determines how quickly does a network update its hyperparameters. High learning rates speed up the training process but can result in gradient convergence becoming more difficult. On the other hand, slower learning rates will ensure a smooth convergence while drastically increasing the time needed to learn.

Activation function

Activation functions ensure that non-linearity is inserted into the problem, otherwise the results would have no interest. These functions work directly with the input values in each node. We describe some commonly used activation functions in subsection 3.2.4.

Loss function

Another important function in backpropagation algorithms is the loss, error or cost function, which determines the difference between a model's projection and the actual result. There are many examples of loss functions that can be used, with some popular ones being the Mean Squared Error (MSE) [WB09], which measures the average of the error squares (Equation 3.7), or the Binary Cross-Entropy [DBKMR05], used in binary classification problems with two outcomes (Equation 3.8). Loss functions can fall into three categories: regressive, classification and embedding.

$$MSE = \frac{1}{n}\sum_{t=1}^{n} e_t^2 \tag{3.7}$$

$$BinaryCrossEntropy = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{K} t_{i,j}\log(p_{i,j}) \tag{3.8}$$

Other hyperparameters

There are other hyperparameters besides the ones already listed, such as the number of epochs, the minibatch size and the momentum value. An epoch is defined as one total pass of the training data through the network. The minibatch size is the number of samples received by the network before the parameters are updated. Finally, the momentum value prevents high swingings and allows the model to decide which direction to go next based on the knowledge of previous steps.

3.2.4 Activation Functions

An activation function exists inside each node and works with its inputs and associated weights to calculate its outputs which will serve as inputs to the next layers. A simple function can be linear or polynomial of one degree, although linearity does not provide any insightful knowledge into

the data because most real-world situations are non-linear in nature. In this section, we present four of the most commonly used activation functions in ANN and DL problems.

Sigmoid or Logistic

$$f(x) = \frac{1}{1+e^{-x}} \tag{3.9}$$

The sigmoid or logistic activation function, which can be seen on Equation 3.9, is one of the simplest activation functions to understand, can be successfully applied to most problems and can be used for probabilistic calculations due to it ranging from 0 to 1. There are however some downsides to it, such as having to deal with vanishing gradient issues. It can also be harder to optimize and can have slow convergence times due to exponentiation [MDKF08].

Hyperbolic Tangent

$$f(x) = \frac{1+e^{-2x}}{1-e^{-2x}}, x \neq 0 \tag{3.10}$$

Tanh, or hyperbolic tangent, is also a very common activation function and works very similarly to sigmoid (Equation 3.10). Because this function ranges from -1 to 1 instead of 0 to 1, optimization is easier compared, but tanh still suffers from much of the same problems as sigmoid, such as exponentiation and vanishing gradient issues [KK92]. A graph comparing both Sigmoid and Tanh can be seen on Figure 3.9 [10].

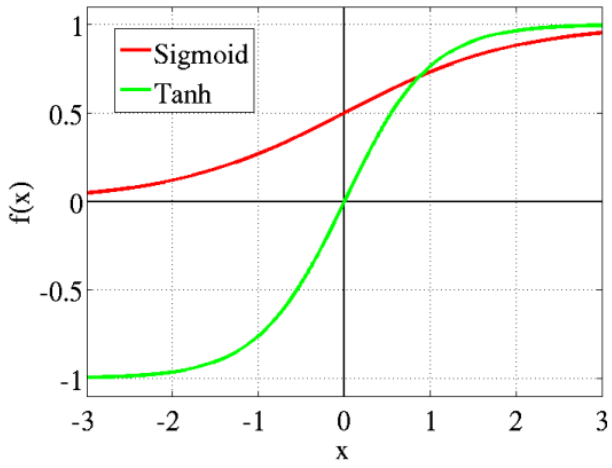

Figure 3.9: Sigmoid and Tanh

[10] Image source: https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6

Rectified Linear Unit

$$f(x) = \max(0, x) \tag{3.11}$$

The Rectified Linear Unit (ReLU) has grown in popularity over the years and is today one of the most used activation functions in the field [NH10]. It ranges from 0 to infinity, transforming every negative value in 0 and maintaining all the positive values (Equation 3.11). It also avoids the common issues of vanishing gradient and exponentiation present in both hyperbolic tangents and sigmoid functions. By discarding negative values however, a heavily dependent problem on negative inputs may not have the correct mapping of these values. This can easily be fixed by knowing this knowledge in advance and inverting the input values from positive to negative and vice-versa. Another issue is the decaying gradients that may arise in some nodes, a problem called "dying ReLU". This problem can be fixed by using an alternative function called Leaky ReLU which inserts a variation on the negative side, usually 0.01, providing a chance of recovery from the dying tendency [CUH15].

Softplus

$$f(x) = \ln(1 + e^x) \tag{3.12}$$

Softplus is an approximation of the ReLU activation function (Equation 3.12), which provides smoothness and differentiability near 0 where the ReLU just suddenly changes. Both softplus and ReLU are highly similar otherwise, as can be seen on Figure 3.10 [11]. ReLU is usually the preferred choice though, because it does not have logarithms and exponentiation in its formula making calculations easier [GBB11a]. The derivative of the softplus function is the sigmoid function.

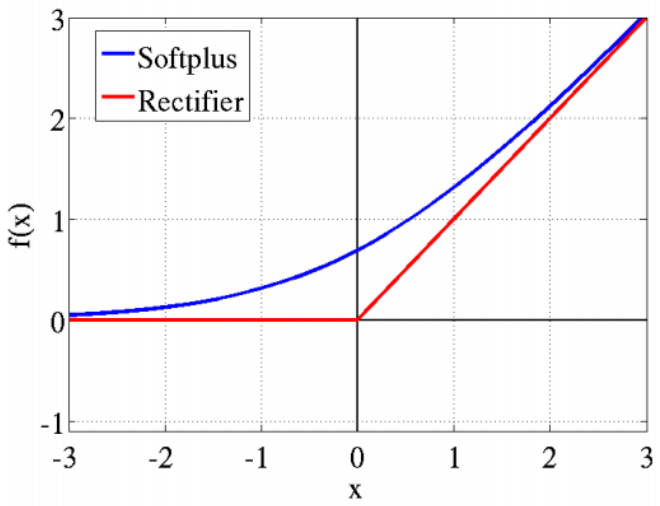

Figure 3.10: Softplus and ReLU

[11] Image source: https://www.quora.com/Why-do-we-not-use-a-differentiable-approximation-of-ReLUs

3.2.5 Architectures

In this section we document five commonly found architectures in DL problems and relate each of them with the concepts approached in previous sections.

Restricted Boltzmann Machines

Restricted Boltzmann Machines (RBMs) are simple networks composed of two layers: the input or visible layer, and the hidden layer. The nodes in each layer link with every node in the other layer, with no connections existing between two nodes in the same layer. Each hidden node receives the input values multiplied by their associated weight, where they will be added up before being passed through the activation function, revealing the output for that particular hidden node. A stacking of several RBMs results in a specific type of architecture called deep-belief networks, which differ from common FFNNs due to their bidirectional network which allows information flowing in both directions [LRB08]. An example is illustrated in Figure 3.11 [12].

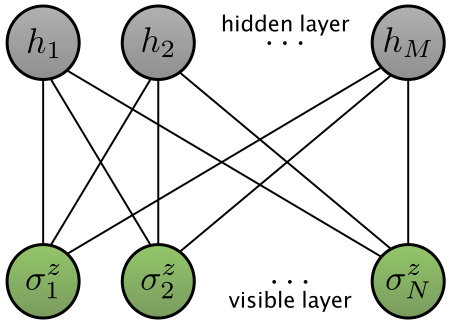

Figure 3.11: Architecture of a restricted boltzmann machine

Convolutional neural networks

Convolutional Neural Networks (CNNs) are a type of deep ANN primarily used in image classification, clustering or pattern recognition problems, although they have also been successfully applied to text and sound classification [KSH12, Kim14]. These networks have applications in the most diverse of real world areas from traffic sign recognition to empowering vision in self-driving cars. LeNet [L+15], a CNN architecture created for simple recognition operations such as reading digits and zip codes, was created around 1990 and was pioneer in deep ANNs, rendering this type of networks as one of the reasons behind DL's popularity growth over the years.

Models based on a CNN architecture can have many different forms, but are usually based on the four main tasks seen on Figure 3.12 [13]. These tasks are:

[12] Image source: https://www.ethz.ch/content/specialinterest/phys/theoretical-physics/cmtm/en/cmt-qo-news/2018/03/reinforcement-learning-of-the-many-body-wave-function-on-rbms.html

[13] Image source: http://www.mdpi.com/1099-4300/19/6/242

- Convolution, consisting in the feature extraction from the input data. This extraction is done by filters or feature detectors that perceive particular conditions (such as edge recognition in the case of images) by scrolling through the data and producing feature maps.
- Introduction of non-linearity functions such as Sigmoid or ReLU after each convolution.
- Sub-sampling or pooling, which is the dimensionality reduction of the feature maps. This results in more manageable data representations and a reduction in the number of parameters, controlling potential overfitting.
- Construction of a fully connected layer, an ANN with a non-linear activation function that uses the sub-sampled results to display a higher level classification of the learned data.

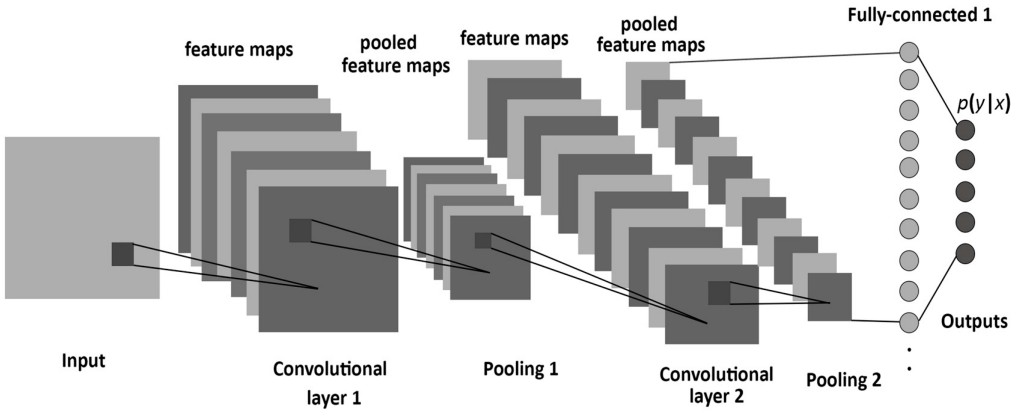

Figure 3.12: Architecture of a convolutional neural network

Recurrent neural networks

Another type of architecture is called Recurrent Neural Network (RNN), and is particularly used in problems with sound or sequential data, such as speech recognition, natural language and sentiment analysis [GMH13]. Figure 3.13 [14] shows a basic RNN architecture. The layer disposition is very similar to FFNNs, with an input layer followed by a hidden layer and an output layer. RNNs are dynamic networks, with their state changing continuously until an equilibrium is reached. They mainly differ from FFNNs because they allow feedback between nodes, with each hidden node's computation being a combined calculation of the input value and the information produced in previous nodes, as can be seen on Figure 3.13. Before being fed to an RNN, the data needs to be pre-processed into vectors using methods called word embeddings [Gro13].

Many variations of RNN exist. Examples include an independently recurrent neural network or IndRNN, a special type of RNN which solves the vanishing gradient issue that usually appears in more basic forms. A long short-term memory or LSTM is another type of RNN that tackles the problem of vanishing gradients by adding elements called "gates". Yet another variation called

[14] Image source: https://towardsdatascience.com/learn-how-recurrent-neural-networks-work-84e975feaaf7

Hopfield network in which all connections between nodes are bidirectional, is used in the studying of many mathematical known problems such as the traveling salesman.

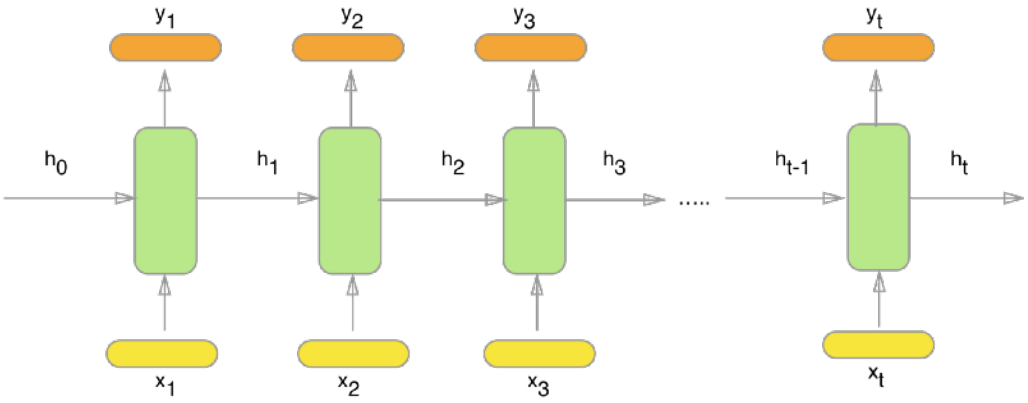

Figure 3.13: Architecture of a recurrent neural network

Autoencoders

Autoencoders are ANNs which are used for feature extraction and dimensionality reduction. They work by compressing the data from the input layer to a hidden layer which usually have less nodes, and then by decompressing that information from the hidden layer to the output layer, usually with the same number of nodes as the input layer. Compressing is also called encoding whereas decompressing is known as decoding. With this approach, an autoencoder aims at learning an abstract representation of the data and possibly other non-linear knowledge that could not be obtained from the raw reads. An example of an autoencoder with six nodes on both the input layer and the output layer and three nodes on the hidden layer can be seen on Figure 3.14 [15].

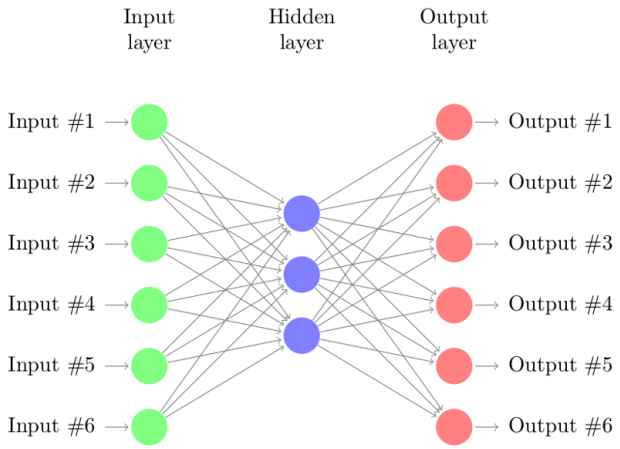

Figure 3.14: Architecture of a simple autoencoder

[15] Image source: https://edouardfouche.com/Neural-based-Outlier-Discovery/

Denoising autoencoders are a special type of autoencoders that introduce randomly corrupted data called noise to the input vales. The goal of this networks is to denoise the corrupted data in the process, attempting to draw insightful information from that reconstruction. Stacked autoencoders (SA) are simple autoencoders stacked on top of each other. By empowering consecutive layers of autoencoders, the input data becomes more and more compact in several smaller steps compared with basic autoencoders where the compacting process is done all at once. This results in a significant loss of information as the training occurs. Denoising autoencoders can also be stacked, resulting in Stacked Denoising Autoencoders (SDAs). SDAs are extensively used in the pre-training process of unsupervised learning problems [Bal12].

Multilayer Perceptrons

MLPs are a type of FFNN with more than two hidden layers. Although FFNs can have only one hidden node and this is usually enough to solve most linear problems, in the context of deep learning problems more than one hidden node is usually considered. It is one of the most common DL architectures, being very easy to understand and delivering state of the art results for a wide array of problems [PM92]. An example of a MLP can be seen in Figure 3.15 [16].

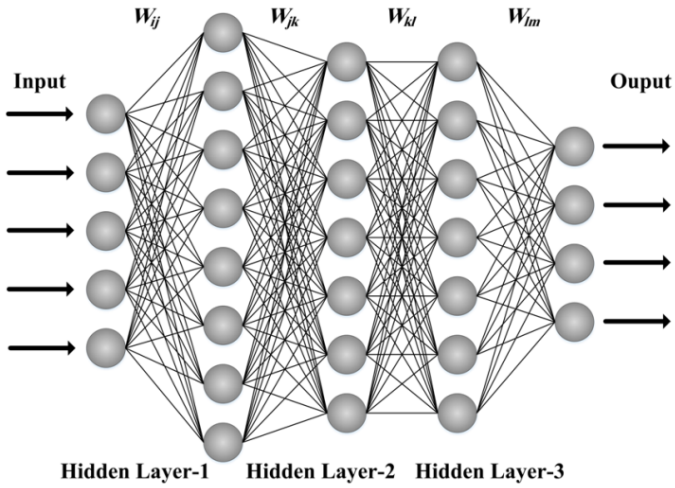

Figure 3.15: Architecture of a multilayer perceptron

3.2.6 Deep Learning Frameworks

Several frameworks, programs or libraries dedicated to DL are available nowadays, and most of them have several distinct features from each other. In this section, we show advantages and disadvantages behind the most popularly used DL frameworks.

[16] Image source: http://pubs.sciepub.com/ajmm/3/3/1/figure/4

TensorFlow

TensorFlow [ABC+16] is a free open-source library developed by Google which besides DL capabilities also offers an extensive math library for other data science purposes. It works with structures called tensors which simulate scalars, vectors and matrices, and provides calculations between these. The main advantages of this tool include intuitive numerical computation, deployment on multiple CPUs or GPUs and a powerful data visualization interface. It proved to be slower than other state of the art frameworks in some benchmarking tests, such as Theano and DL4J [FRA]. Despite this, TensorFlow is one of the most used DL frameworks in the industry and is widely regarded as the cause behind the sudden burst of popularity of DL in machine learning. The framework is cross-platform and provides programming in Python and C++. It is particularly popular with large-scale projects.

Deeplearning4j

Deeplearning4j (DL4J) [DL4] is another framework written in Java which, like the name suggests, was created with the sole purpose of addressing DL problems. It works with most state of the art algorithms in DL, and provides many exclusive features to handle common problems in data mining, such as an embedded class to deal with data pre-processing and another one for feature extraction. The framework is very straightforward and high level, with many shortcuts to frequent actions such as model creation. It distinguishes itself from other frameworks such as TensorFlow which work with lower level numerical calculations. DL4J also allows integration with Spark, a distributed computing library for GPU usage in machine learning. It is completely open source, with all the development being made on GitHub, and has a very committed team of creators behind it constantly updating the framework with more features. There is also a vast collection of examples and models that can be rapidly downloaded and built, to test and explore the main capabilities of the framework.

CAFFE

Convolution Architecture For Feature Extraction or CAFFE [JSD+14] is an open-framework written in Python with interfaces in C++ and MATLAB as well. It provides most of the features found in other state of the art frameworks, and as the name suggests, it's particularly good at handling CNNs and is a popular workspace for visual recognition problems. It is fast, portable, and also allows distributed processing across many computers. The language modeling is poor however, and complex networks such as LSTMs can be quite difficult to implement using a low level approach.

Torch

Torch [IJD] is an open source learning library developed in the Lua programming language and distributed on GitHub. Due to this Lua background, Torch is one of the fastest state of the art frameworks. It is not primarily focused on DL, but can process most of its algorithms once it

installs the *nn* extension package which deals with neural networks mechanisms. Same as TensorFlow, Torch provides numerical calculations through tensors. It is one of the oldest machine learning frameworks with DL capabilities, with the first stable release dating back to 2002. Nowadays, Torch is supported by Facebook which releases frequent extension modules, and is one of the most popular frameworks rivaling Google's TensorFlow.

Apache MXNet

A recent option, Apache MXNet [OIJ] already proved itself to be worthy of attention, being picked by a few online reviewing websites as one of the most promising DL frameworks in the market. Dating back to 2016, MXNet is a flexible and scalable option that gives support to the most commonly used DL algorithms, providing the user the option to construct DL models with the language of their choice. It is written in many languages, including C++, Scala and Perl. The website provides a rich online course where the user can learn how to work with the majority of the framework.

Other available tools

Besides the aforementioned frameworks, there are many other available options which are purely dedicated or allow implementation of DL models. Examples include Keras [T39], another DL-focused tool written in Python which is particularly good in classification problems and belongs to the core API of TensorFlow; Chainer [AJ3], which allows runtime modifications to the network; PlaidML [MIK], which provides support for GPU integration independently of the model; and Microsoft's Cognitive Toolkit [ZUA], which is a recent dedicated framework that gives access to most DL architectures and is distributed within the most recent versions of Microsoft's .NET framework.F

3.3 Chapter Summary

In this chapter, we explained all the necessary data mining and deep learning concepts to understand the scope of our project. We also presented the most popular data mining tools and discussed their differences, advantages, disadvantages and their market share. We ended this chapter by exploring the main features behind a few state of the art frameworks in DL.

Chapter 4

Development and Evaluation

In this chapter, we describe the methodology followed during the enactment of this project. We then proceed to a thorough explanation on the implementation of the solution and an analysis on the obtained results.

4.1 Problem Analysis

We start by giving an overview on the general problems encountered during the testing of a state of the art gene prediction tool. We then proceed to an analysis on alternative solutions using more recent technologies such as DL and how we can attain the proposed results, followed by a discussion on the most expected difficulties and also ways devised to overcome them. We end this section by explaining the working environment that was used to implement the results.

4.1.1 Empirical Testing

DNA has always puzzled scientists. Ever since the first genome sequencings, attempts have been made in order to understand how does it work and what is the utility of all its regions, particularly the protein-coding sequences called genes. The annotation of genes is a major step after the sequencing of a specific genome, consisting in the identification of portions in the genome with any biological significance. Done using a variety of steps, purposes and tools, one important phase in gene annotation is gene prediction.

Computational tools have been developed over the years which ease the cumbersome task of gene prediction. These tools (some of them described in Chapter 3), are usually specific to a certain organism. While the more classic tools provide good results and find the most probable exons and sub-exons in large genomic sequences, there are still several limitations in the predictions and behaviors of most current tools.

Fourth-generation software such as AUGUSTUS and GENSCAN are based on GHMMs or Bayesian methods as the type of algorithms behind their implementations. Although these methods have shown good results in this class of problems, since most knowledge about genes and their structure is still limited, HMMs results are often flawed or inaccurate.

In order to access and evaluate both the behavior and the metrics of fourth-generation gene prediction tools, we made two different experiments. We used the GENSCAN software tool, which is one of the most accurate fourth-generation tools and has the advantage of having an online interface without the requirement of additional downloads or dependencies. We also wanted to investigate Augustus, a similar tool with a different background, but it was unavailable at the time of these tests.

In the first test, we selected a large DNA sequence for testing. This sequence was the primary assembling of the third human chromosome retrieved as a FASTA file from the GenBank repository [1]. Since GENSCAN only accepts sequences up to a million base-pairs in length, we had to select a subsequence from the approximately 200 million that make up the third chromosome. After serving the 999956-character long subsequence as an input, GENSCAN predicted a total of 28 genes scattered across 143 different exons. Along with other information such as an indication of the starting and ending positions, there was also a probability of that sequence being, in fact, an exon. We found the mean of those probabilities being 0.83 after string manipulation of the parsed results in Java, which is about the same value of specificity calculated in [WZ04] for GENSCAN.

The second experiment consisted on testing individual gene sequences. We wanted to access if GENSCAN could identify homologue genes from *Arabidopsis thaliana*, a species whose samples were used during training. For that purpose, we used four homologue genes retrieved from an online tool called HOVERGEN [2]. We also queried GenBank for four additional human genes from the seventh chromosome, two of them uncharacterized. The results are presented in Table 4.1. The length is the overall size of the FASTA file, and the sequence includes intronic and splice regions as well as other adjacent nucleotides besides exons.

Table 4.1: Genes selected for testing. The first four are homologues.

Gene ID	Official designation	Length	Result (%)
COPS6	COP9 signalosome complex subunit 6	3287	Not found
PCOLCE	procollagen C-endopeptidase enhancer	6001	1 gene, 11 exons (0.99)
MDH2	malate dehydrogenase 2	19770	Not found
TRIM74	tripartite motif containing 74	51819	4 genes, 25 exons (0.76)
STYXL1	serine/threonine/tyrosine interacting like 1	54289	2 genes, 14 exons (0.56)
MYL7	myosin light chain 7	2566	1 gene, 7 exons (0.72)
-	uncharacterized LOC105375497	9836	1 gene, 10 exons (0.73)
-	uncharacterized LOC107986763	26107	2 genes, 8 exons (0.37)

The results show that GENSCAN is still state of the art when trying to predict genes, even if the probabilities aren't always high. All the predictions were calculated almost immediately. The tool failed to predict two of the four homologue genes and identified the TRIM74 gene, the second

[1] Link to FASTA: http://goo.gl/Cjsz3G
[2] HOVERGEN: http://pbil.univ-lyon1.fr/databases/hovergen.php

longest of all the tested sequences, as four distinct genes scattered across 25 exons, instead of a very long gene.

4.1.2 Different Approach

For this project, we wanted to use a different approach on the problem by switching from the markov models algorithms (HMMs, GHMMs and IMMs) currently in use in the majority of gene prediction software to algorithms based on neural networks. According to [VBY+95], neural networks compared to HMMs prove to be more efficient in terms of computational requirements. Another studying proved that ANNs are particularly better than HMMs at handling large amounts of data [PC16], which is usually the case in biological problems. Deep learning algorithms in particular are outperforming other classifiers and have been the state of the art approach in the resolution of many problems [GBB11b, KLSS17, LS17].

On the most basic level of every gene prediction algorithm there is a classification problem. Before the eventual exploration and annotation, a gene must be detected as such, in a environment where protein-coding sequences are not only a clear minority but also harder to detect correctly due to introns and alternative splicing. Our contribution is the studying of how large DNA sequences behave with DL techniques. To achieve that, we selected a MLP as our architecture and trained it with the intention of classifying input sequences as either genes or not. By doing this experiments, we want to promote future implementations of state of the art gene prediction software that could be trained in a similar fashion to a more demanding purpose.

4.1.3 Challenges

In order to accomplish our goals, we had to address some initial challenges related with the project and the concepts behind it.

The first and the most important problem is the creation of a proper dataset for training and testing, both in size as in content. The size reflects the fact that deep learning algorithms are usually demanding of large amounts of data to achieve state of the art results. This can be seen in [Cho17], where a CNN for image classification was trained with a dataset of 350 million images and 17,000 classes. In one particular experiment, the algorithm ran in a distributed system across 60 GPUs and took up to a month to finish training. Time and computational power is usually among the constraints when working with this type of problems, and the size and complexity of a dataset is paramount in determining how much of those variables is needed. The content is equally important. Since the algorithms can only extract features from the data that is provided to it, we need to select a collection of sequences which correctly capture the context of the problem, namely the various components of the genome. This filtering of adequate data is usually done during the pre-processing phase of the implementation.

Another challenge that usually appears after the data preparation is completed is the suitable tuning of the chosen DL architecture. This encompasses the correct choice of hyperparameters and other variables related to the model. Since most hyperparameters are dependent on the problem's

context and goals and because doing the adequate adjustments can result in the training process occurring all over again, we need to study beforehand the available options through research and experiments before actively pursuing the outlined goals.

4.1.4 Workspace

We experienced some limitations regarding the computational needs to our work. Although most DL projects that deliver state of the art results do so by working with connected computers in a process called cluster-computing, we had a hard time finding other machines to add to our environment. The work was ultimately developed using a single computer running Microsoft Windows as the operating system, with additional details present in Table 4.2.

Table 4.2: Working environment

Computer Specifications	
CPU	Intel®Core™ i7-4700HQ Quad Core
Clock rate	2.4 GHz
GPU	NVIDIA® GeForce™ GTX 850M
GPU memory	4GB GDDR5
RAM (memory)	16 GB
RAM (type)	DDR4 2x8
Operating system	Windows 10 64-bit
Software and Tools	
Java	8.0
IntelliJ IDEA	2018.1.5
DL4J / ND4J / DataVec	0.9.0
Maven	3.5.3
JDMP	0.3.0

Java was the programming language of choice in the development of this project. The reasons behind this decision were the greater performance on large-scale systems compared to languages such as R and Python [Lan] and the usage of DL4J as the deep learning framework to build our model. We also used Java Data Mining Package (JDMP) [3], a Java library for machine learning and big data analytics, to visualize the raw data during the pre-processing phase. Our first intention was to use RapidMiner due to its great capabilities on data mining, but we ended up using an available library within the already chosen solution stack.

As mentioned, we created our model using the DL4J framework. This decision was made due to several reasons, such as the high-level purpose of the language, easy training with GPUs and a vast and comprehensive documentation with real examples. We also used two of its extensions:

[3] https://jdmp.org/

DataVec [4] for data iteration and normalization and ND4J [5] for k-fold cross validation and feature reduction with PCA.

IntelliJ IDEA from JetBrains [6], with dependency management and automated build guaranteed by Apache Maven [7], was the integrated development environment of choice. The versions used in both the software and the development kits at the final stage of the implementation are documented in Table 4.2.

4.2 Dataset Preparation

The first step was to arrange a proper dataset that would represent the context we were working on. Since we have a classification problem with two classes, we needed to select a good and balanced number of samples from both classes. One common problem in machine learning problems is to determine how large should be this number. According to [Pla], there is usually no right answer and the number depends on many factors including how different are the classes, while in [KVJK05] we can see that differences between coding and non-coding regions may be hard to identify. Based on these two factors, we decided to construct a dataset of 50000 samples from the human chromosome, with 25000 being coding regions and 25000 being non-coding regions.

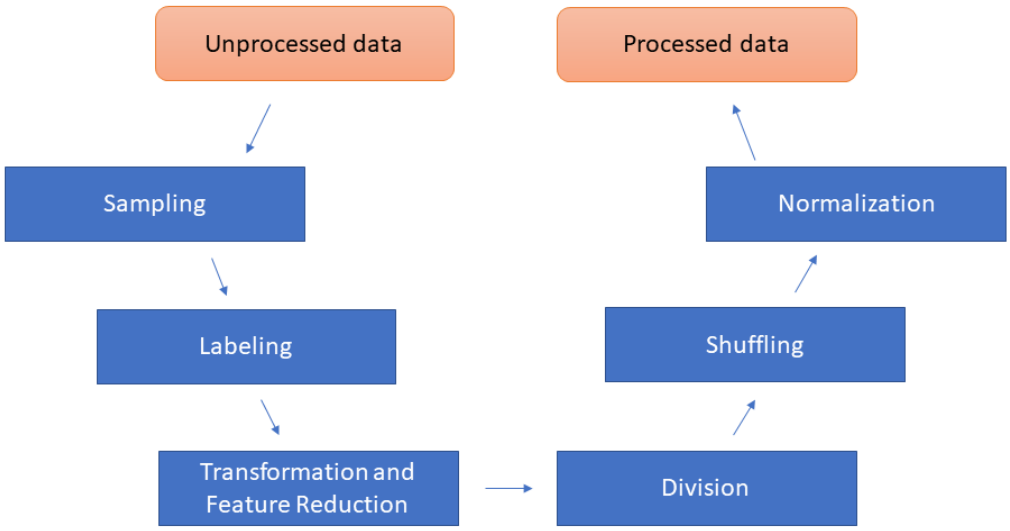

Figure 4.1: From unprocessed data to processed data

Our raw samples were obtained from two different sources. GenCode, the first source, provided two files while Noncode, the other source, provided one file. Although in separate files, both coding and non-coding regions were retrieved from the same transcript sequences on the referred chromosomes by GenCode. Since there was not enough non-coding sequences in the file from GenCode, we chose Noncode to download additional data. The three files were in FASTA

[4] https://github.com/deeplearning4j/deeplearning4j/tree/master/datavec
[5] https://github.com/deeplearning4j/nd4j
[6] https://www.jetbrains.com/idea/
[7] https://maven.apache.org/

format (extension *.fa*). The data was subsequently pre-processed following the steps presented in Figure 4.1.

4.2.1 Sampling

The first step after downloading the files was to select which 25000 coding regions and 25000 non-coding regions would be used for the rest of the process. The file with the protein-coding sequences had 203835 sequences in total, but only 82335 of those sequences were genes. The GenCode file with non-coding sequences had 35632 samples and the Noncode file had an additional 200000 samples. An example of a raw sample can be seen in Listing 4.1.

```
1  >ENST00000410048.5|ENSG00000186973.10|OTTHUMG00000007286.4|OTTHUMT00000019026.2|RP5
       -1034F7.1-003|FAM183A|500|protein_coding|
2  CAGCCAGCCTGCATGAGGTTCTCTACGTTTGAGACTGGCTTCTCCACGGAACTTGGTCTG
3  TAGCCACAGAGCAAACTGCAGGGCCATTGACAGAGAATGAAGAGGAGAGTGAAGAGTGGA
4  ACACATCTAATGTGTTCTCTGGAGGTTCATACAGTCACCAGGAAGCCCATGTCTTGGCAT
5  GATAACCTGGAGGAACCTGCAGATGCCAGGTTTCTGAATCTCATTCACCATGCTGCCCAG
6  GGACCAAGGAAGAAGTACCCAGAGACACAGACTGAAAACCAGGAAGTTGGATGGGACTTA
7  GAGCCCTTGATCAACCCAGAACGCCATGACCGCAGGCTGAATCACTTCAGGGTCTGCAGT
8  GACATCACTCTGTACAAGGCTAAAACGTGGGGCTTAGGAGATGATCACCACAAGTAGCAT
9  CCCAGCGGATGAGCCCATCTGTGGATCTAATGCCTTAAGTGTGCACAGCCCAGAGAAATA
10 AAATACTACTTTAAAACGAA
```

Listing 4.1: Example of an unprocessed sample. All the samples begin with > and have a description, the size and a type (a protein coding sequence with 500 nucleotides in this case).

Although some models in deep learning are designed to work well with sequences of varying length [SFWS18], for simplicity and due to our proposed architecture's nature, we decided to work with samples with the same size. Since there were only 48 protein-coding samples with exactly 500 nucleotides, we decided to choose larger sequences and dispose of additional nucleotides on both ends. Called truncating, this method is usually considered a risk since we may be losing valuable information. In this case however, we would be disposing of flanking regions [8] either with little connection to the actual translating region of the sequence or with no connection at all.

Using regular expressions, we found 25121 matches of protein-coding sequences from 500 to 599 nucleotides in length, just enough to build one part of our dataset. Starting from samples with 501 length, we truncated the excess number of nucleotides on both ends as can be seen on Figure 4.2.

We have applied the same algorithm to non-coding regions, although due to the higher number of available sequences compared with protein-coding, we ended up truncating much less nucleotides in total. Besides the selected samples, we eliminated every other element in the file, including the initial information about the sequence. We also copied non-coding regions from

[8] Regions delimiting a gene that are not translated

the Nocode file into the Gencode one. After this phase we were left with two files, each with 25000 sequences of 500 nucleotides in length. The sequences were separated by a new line escape character ($\backslash n$).

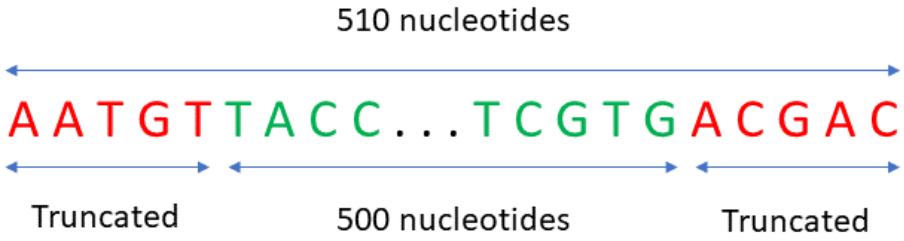

Figure 4.2: Example of truncating a sequence

4.2.2 Labeling

After transforming the data, we associated each sequence with its class. 1 was attached after each coding sequence and 0 after each non-coding sequence.

4.2.3 Transformation and Feature Reduction

We have transformed our data both manually as well as through a feature reduction algorithm.

Our manual transformation consisted on changing the data type and presentation. Since deep learning models can only work with numerical values, we needed to transform the characters into numbers. Instead of replacing each letter with a number, we based our approach on [NTN+16] and on the genetically importance of three nucleotide sequences [PRBB17].

By sliding through each sequence, we replaced every three joined characters with an associated number and a new line. This resulted in a total of 64 numbers, one for each of the 4 x 4 x 4 possible combinations of three nucleotides. An example of our data transformation can be seen in Figure 4.3.

Although the data transformation in Figure 4.3 is enough for computational interpretation, the data could still be considered raw in some ways, especially the dimension. As it was right now, our data set could be seen as a matrix with 50000 rows and 498 columns. Since we wanted to improve the performance of our model by working with an optimized dataset, we needed to conduct some form of dimensionality reduction. PCA is a simple and common algorithm for feature reduction. It works by finding patterns in data and correlations between the variables through the use of two concepts called eigenvectors and eigenvalues. After discarding the values considered the least valuable, the algorithm projects the remaining data onto a smaller subspace without losing much information.

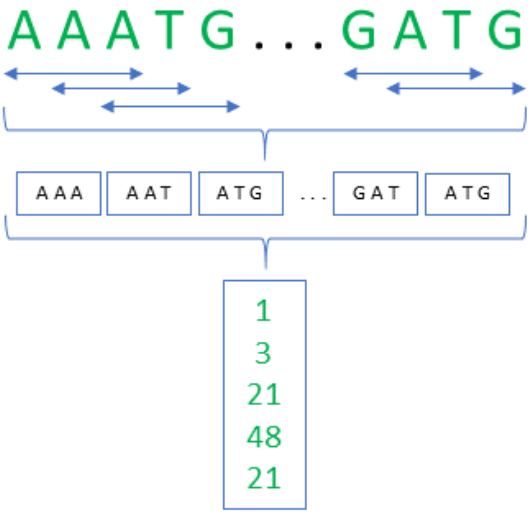

Figure 4.3: Example of transforming a sequence

We used an extension of DL4J called ND4J which provides a straightforward implementation of PCA [9]. We first converted our data so that it could be interpreted by *INDArrays*, the structure behind ND4J that represents an n-dimensional array. We then chose the number of features to be preserved in the lower-dimensional projection. This number was upper bounded by the original number of features, 498. To decide this number, we needed to be sure that by removing certain features we were not compromising much information. To asses this, we applied the following algorithm:

- We placed the eigenvalues in ascending order. This was done in order to decide which ones to exclude first, since the lowest eigenvalues represent the least information about the distribution of the data.
- We iteratively divided the sum of the **K** lowest eigenvalues by the sum of all the eigenvalues. This represented the ratio of lost information, and our goal was to minimize this ratio.

After several iterations, we concluded that the first 88 lowest eigenvalues represented a total loss of 2% of the information, which seemed like a good number. We then excluded those, and projected the remaining eigenvectors onto a 50000 x 410 new dataset.

4.2.4 Division and Shuffling

To train and test our model, we used a 5-fold cross-validation method to evade common issues in data mining such as overfitting and lack of variance. With this technique, in every fold the dataset is divided into two sections: one training set with 80% of the total data and one testing set with the remaining 20%. We joined the protein-coding regions with the non-coding regions beforehand and

[9] https://github.com/deeplearning4j/nd4j/blob/master/nd4j-backends/nd4j-api-parent/nd4j-api/src/main/java/org/nd4j/linalg/dimensionalityreduction/PCA.java

did a random shuffle of the entire dataset. This process resulted in a training set of 40000 samples and a testing set of 10000 samples which changed five times throughout a full experiment. This method is illustrated in Figure 4.4.

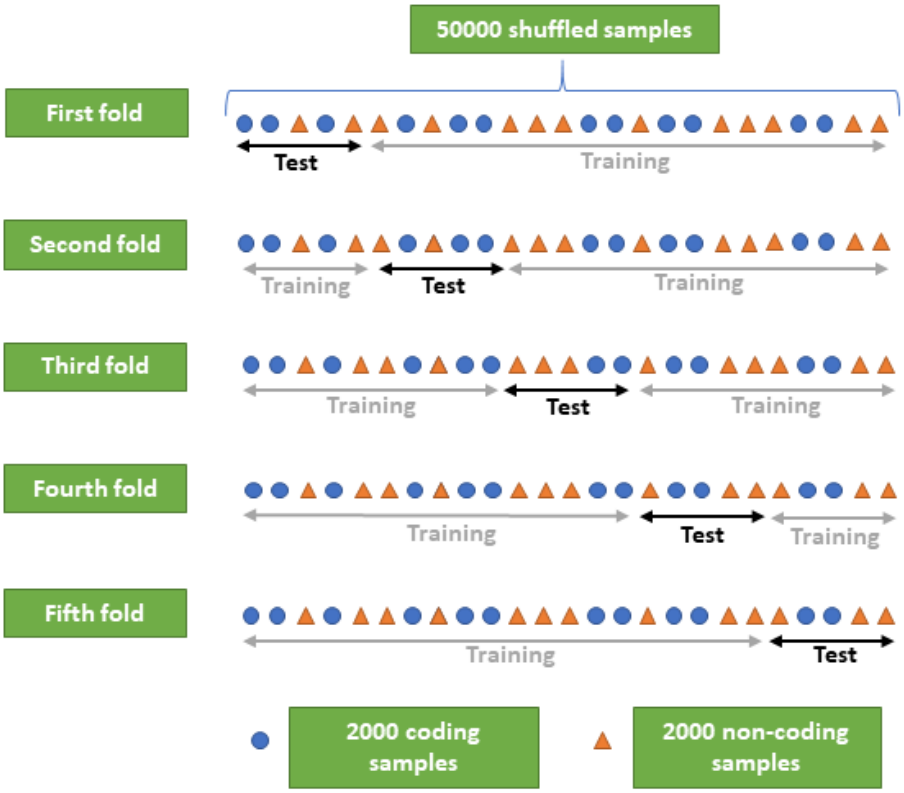

Figure 4.4: Our project's 5-fold cross-validation

4.2.5 Normalization

A common practice to improve efficiency is to normalize the input data. This was achieved by using a class provided by DataVec that normalizes feature and label values to have 0 mean and a standard deviation of 1. As a final step, we also automatically created two directories where we stored 50000 converted CSV files in each of them: one with all the sequences and another one with all the associated labels. This was done for two reasons. One was to simplify data visualization by JDMP, which works better with CSV files. The second reason was to physically store the processed data in order to avoid all this pre-processing every time we ran our algorithms.

4.3 Implementation and Results

In this section, we explain the course of our development phase after creating the dataset. We report how we created, configured and trained our MLP using DL4J, and which experiments were done to access their metrics and other performance measurements. We end the section by comparing and discussion the results.

4.3.1 Multilayer Perceptron Model

We considered a MLP over other DL models for a number of reasons, particularly its plainness and a well documented good performance on supervised learning and classification problems [GD98]. It is also usually easy to configure and adjust. Moreover, although other architectures described in Section 3.2.5 have been delivering state of the art results over the past few years, they require very demanding computing power to do so and are often hard to tune. An abstraction of our implemented architecture can be seen on Figure 4.5.

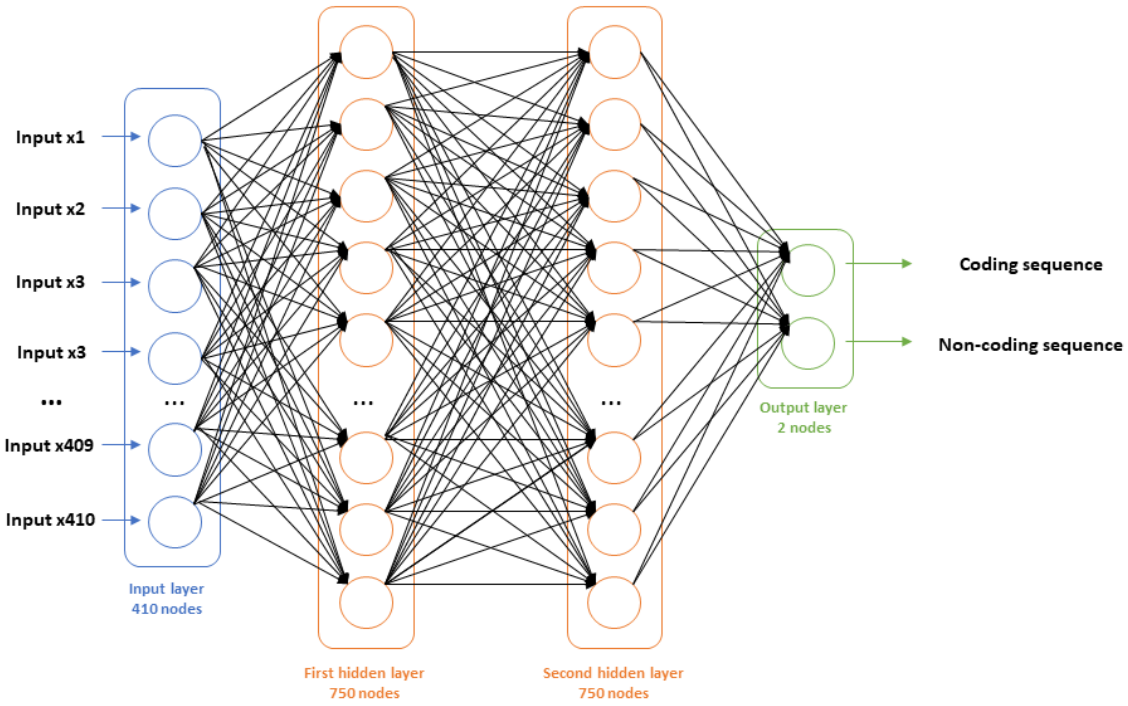

Figure 4.5: Developed MLP's architecture

The input layer has 410 nodes corresponding to each value in the transformed data. For experiments with the data unprocessed by PCA, we have also used an input layer with 498 nodes. These values enter the network through the use of iterators provided by DataVec, which scrolls through all the processed data and converts it to structures that can be understood by DL4J models. The values are propagated to two further hidden layers, each with 750 nodes each. Finally, these values converge into an output layer consisting of two nodes that classify the input DNA sequence.

Common MLPs and ANNs in general have exactly one input and one output layer. The number of nodes in the input and output layers is also relatively easy to identify and are dependent on the context. The same does not happen however with the number of hidden layers and the number of nodes in each of those layers. Opinions are not consensual [Sta09], with some arguing that hidden layers should have more nodes than the input layer and others arguing the contrary. The total number of hidden layers is also subject to discussion, but more than one is usually plausible for smaller datasets [Hin92] or overall complex in nature, such as ours. Based on this research, we ended up selecting a larger number of nodes in the hidden layer compared to the input layer.

Configuration and hyperparameters

Configuring the model's hyperparameters is one of the most important tasks in any DL problem. It is also known to be one of the hardest, since only a few choices are usually the norm in most implementations [DSH13]. The vast majority of hyperparameters are dependent on the type and context of the problem and need to be empirically tested, sometimes in a trial and error approach.

In DL4J, the models are built using the MultiLayerConfiguration and NeuralNetConfiguration classes. Besides the already presented configuration of the layers, Table 4.3 presents the other selected hyperparameters for this MLP.

Table 4.3: MLP's hyperparameters

Hyperparameters	
Seed	150
Weight Initialization	Xavier
Activation Functions	LeakyReLU, Sigmoid
Updater	ADAM
Learning Rate	0.001
Backpropagation	true
Backpropagation Method	Mini-batch gradient descent
Mini-batch	16
Loss Function	Cross Entropy
Regularization	true
Dropout	15%
Pretrain	false
Number of epochs	40
Visualization	5

The hyperparameters used in the end were selected after extensive testing where most of them were experimented with other values. Running the dataset with feature reduction, this configuration resulted in an average running time of 1.5 minutes per fold, or about 6 minutes in total after 5 folds. Without feature reduction, the algorithm took longer to finish: 2 minutes after one fold or 10 minutes after 5 folds. The reasoning behind the choice of the hyperparameters is the following:

- The seed was selected as a random value for reproducibility in the initialization of the weights.
- The weights themselves were initialized using Xavier, a method proposed by Xavier Glorot and Yoshua Bengio in [GB10]. This technique fixes the problem of weights initializing either too large or too small, which may result in the weights becoming useless over time. Xavier's formula makes sure the weights initialize in a sensible range of values through the layers. We also tested initially a normal/gaussian distribution, but the results were poorer.

- LeakyReLU was the activation of choice for all layers except the output one. ReLU in general has been consistently achieving more state of the art results in DL and MLPs problems [XWCL15]. It is also computational cheaper to backpropagate comparing to other commonly used functions such as Tanh and Sigmoid, mostly due to its very simple and linear formula. To avoid the vanishing gradient problem, we chose the alternative LeakyReLU. On the other hand, Sigmoid was used on the output layer since it is the usual activation functions in binary classification problems such as this one.

- ADAM [KB14] was chosen after testing with three other updaters supported by DL4J: Nesterovs, RmsProp and AdaGrad. ADAM trials increased the convergence time and gave better results maintaining the other parameters unaltered. It worked with the default values: alpha as 0.001, beta1 as 0.9, beta2 as 0.999 and epsilon as 10E-8.

- Learning rate is highly dependent on the model, and we couldn't know beforehand which value to use. Usually smaller numbers give greater results, but the model can take much time to converge. We used values such as 0.1 and 0.01, but the results were lackluster. Since we had computational limitations, we ended up using 0.001.

- Backpropagation was done with a default mini-batch gradient descent. DL4J also supported stochastic gradient descents and other rare gradient optimizers such as hessian free, but DL4J recommends mini-batch optimizers over other options due to it being more computationally efficient in their implementations. To use this optimization, we also needed to select the size of mini-batch and pass it over to the configuration class. We chose 16, but any other power of two starting at 16 (32, 64, 128) could have been chosen at the expense of longer convergence times and probably very similar results [bat]. An additional backpropagation boolean flag had to be set as true.

- DL4J provides many well known loss functions and even allows the user to customize their own. We ended up selecting the Cross-Entropy for binary classification as our loss function which usually goes together with many DL problems that use Sigmoid at the output layer [Tan13].

- Dropout is a regularization method that was created to prevent overfitting in models with smaller datasets, such as ours [SHK+14]. This is done by disallowing units from adapting too much. We used a percentage of 15%, since smaller percentages could have minimal results while higher percentages could end up in under-learning. Another additional boolean flag had to be set as true as well.

- Although not directly related with the configuration of the model, we also needed to set the total number of epochs. Besides the algorithm taking longer to finish, the number of epochs needs to be in the optimal range, since larger or smaller numbers can result in over or underfitting. We chose the value 40 after also testing 30 and 50 which resulted in poorer

evaluation metrics. Since we ran a 5-fold cross-validation method, the dataset fully passed throughout the network a total of 200 times.

- The visualization number represented the period of waiting before the loss function would print a result to the screen. This was made so that we could iteratively check the progress of the training.
- An additional but mandatory parameter in DL4J regards pre-training. In unsupervised training for networks like RBMs and Autoencoders, pre-training is usually considered to deliver more state of the art results. Since this is not the case for MLPs or FFNNs in general, this flag was set to false.

4.3.2 Experiments

DL4J provides functions and classes to evaluate a model's performance. For our model in particular, we wanted to access the confusion matrix, the accuracy, precision, recall, F1 Scores and the negative predictive value. We also kept track of the rates of false positives and false negatives. Since we also wanted to assess the performance improvements in an optimized dataset, we have done experiments with the data before and after feature reduction with PCA.

Since the beginning of the project, we iteratively configured and trained the model by making small changes in the hyperparameters, with the evaluation metrics and the confusion matrix being printed in the screen after each fold. After several experiments, the configuration at Table 4.3 delivered the best overall results. These results are documented from Table 4.4 to Table 4.13. Finally, Table 4.14 compares both approaches by placing the averages of all the tables into perspective.

Before feature reduction

Table 4.4: MLP - Metrics after the first fold without PCA

	Confusion Matrix	
	Predicted coding regions	Predicted non-coding regions
Real coding regions	4328	570
Real non-coding regions	1043	4059
Measures		
Accuracy		0.84
Precision		0.81
Recall		0.89
F1 Score		0.84
Negative Predictive Value		0.88
False Positive Rate		0.20
False Negative Rate		0.12

Table 4.5: MLP - Metrics after the second fold without PCA

	Confusion Matrix	
	Predicted coding regions	Predicted non-coding regions
Real coding regions	4517	474
Real non-coding regions	985	4024
Measures		
Accuracy		0.85
Precision		0.82
Recall		0.91
F1 Score		0.86
Negative Predictive Value		0.89
False Positive Rate		0.20
False Negative Rate		0.10

Table 4.6: MLP - Metrics after the third fold without PCA

	Confusion Matrix	
	Predicted coding regions	Predicted non-coding regions
Real coding regions	4585	482
Real non-coding regions	1015	3918
Measures		
Accuracy		0.85
Precision		0.82
Recall		0.90
F1 Score		0.86
Negative Predictive Value		0.89
False Positive Rate		0.21
False Negative Rate		0.10

Table 4.7: MLP - Metrics after the fourth fold without PCA

	Confusion Matrix	
	Predicted coding regions	Predicted non-coding regions
Real coding regions	4395	614
Real non-coding regions	1061	3930
Measures		
Accuracy		0.83
Precision		0.81
Recall		0.88
F1 Score		0.84
Negative Predictive Value		0.86
False Positive Rate		0.21
False Negative Rate		0.12

Table 4.8: MLP - Metrics after the fifth fold without PCA

	Confusion Matrix	
	Predicted coding regions	Predicted non-coding regions
Real coding regions	4448	587
Real non-coding regions	1003	3962
Measures		
Accuracy		0.84
Precision		0.82
Recall		0.88
F1 Score		0.85
Negative Predictive Value		0.87
False Positive Rate		0.20
False Negative Rate		0.12

After feature reduction

Table 4.9: MLP - Metrics after the first fold with PCA

	Confusion Matrix	
	Predicted coding regions	Predicted non-coding regions
Real coding regions	4754	250
Real non-coding regions	498	4498
Measures		
Accuracy		0.93
Precision		0.91
Recall		0.95
F1 Score		0.93
Negative Predictive Value		0.95
False Positive Rate		0.10
False Negative Rate		0.05

Table 4.10: MLP - Metrics after the second fold with PCA

	Confusion Matrix	
	Predicted coding regions	Predicted non-coding regions
Real coding regions	4775	238
Real non-coding regions	469	4518
Measures		
Accuracy		0.93
Precision		0.91
Recall		0.96
F1 Score		0.93
Negative Predictive Value		0.95
False Positive Rate		0.09
False Negative Rate		0.05

Table 4.11: MLP - Metrics after the third fold with PCA

	Confusion Matrix	
	Predicted coding regions	Predicted non-coding regions
Real coding regions	4710	239
Real non-coding regions	467	4584
Measures		
Accuracy		0.93
Precision		0.91
Recall		0.95
F1 Score		0.93
Negative Predictive Value		0.95
False Positive Rate		0.09
False Negative Rate		0.05

Table 4.12: MLP - Metrics after the fourth fold with PCA

	Confusion Matrix	
	Predicted coding regions	Predicted non-coding regions
Real coding regions	4815	227
Real non-coding regions	564	4394
Measures		
Accuracy		0.92
Precision		0.90
Recall		0.96
F1 Score		0.92
Negative Predictive Value		0.95
False Positive Rate		0.11
False Negative Rate		0.04

Table 4.13: MLP - Metrics after the fifth fold with PCA

	Confusion Matrix	
	Predicted coding regions	Predicted non-coding regions
Real coding regions	4749	243
Real non-coding regions	481	4527
Measures		
Accuracy		0.93
Precision		0.91
Recall		0.95
F1 Score		0.93
Negative Predictive Value		0.95
False Positive Rate		0.09
False Negative Rate		0.05

Comparison between the averages of all the tables

Table 4.14: Comparison between both approaches after calculating the cross-validation average

Evaluation Measure	Without PCA	With PCA
Accuracy	0.84	0.93
Precision	0.82	0.91
Recall	0.89	0.95
F1 Score	0.85	0.93
Negative Predictive Value	0.88	0.95
False Positive Rate	0.20	0.10
False Negative Rate	0.11	0.05

4.3.3 Assessment and Discussion of Results

By analyzing our experiments, we conclude that using PCA delivers the best overall results compared with the raw data. This can be seen not only in the achieved accuracies, precisions and other calculated metrics, but also in the considerable reduction in the running time: from 10 minutes with an unprocessed dataset to almost 6 minutes with PCA. This is the expected behavior, since by reducing the dimensionality of the problem we have lesser nodes and therefore less calculations to handle.

Approximately 93% was the best accuracy we could attain. Although this is not perfect, we believe we couldn't achieve much higher results due to the complex nature of the genome. Even though genetic sequences usually share some similarities, there might be some sequences that are just completely different from the rest and the algorithm could not correctly interpret them.

Another possible reason for this was the truncating at the pre-processing phase that could have resulted in important signals being removed in some cases. Moreover, deep learning algorithms are known to provide greater results with very big datasets which was not our case.

Another noticeable result was the fact that non-coding regions were wrongly predicted more times than coding regions in all the cases, even after feature reduction by PCA. This could be due to a number of reasons, but in the context of our dataset we believe it happened because we didn't specify what type of non-coding sequence we were working on. All the 25000 non-coding sequences were retrieved randomly by their size only, with their types being disregarded. Since some non-coding elements that bear resemblance with genes (such as pseudogenes) were present in the original files in high quantities, some of them probably ended up in the dataset and were classified as protein-coding by the model.

As a final assessment in our work, we wanted to test some sequences in GENSCAN to check the results. The goal was to understand why some non-coding sequences were misclassified by the model. We didn't test coding regions because they had already been tested in the empirical usage of gene prediction tools on Section 4.1.1. We have also done this on a unprocessed dataset by PCA, due to the fact that PCA completely transforms data and we could not chase the sample back to its original form. To do this experiment, we first started by extracting the TNs and the FPs after the model finished its fifth fold evaluating. We then systematically searched for pseudogenes in the FP set through the corresponding sequences in the original files. This had to be done manually and took some time, but eventually we found two pseudogene sequences. After selecting two other random ones from the TN set just for the sake of comparing, we provided the four genes as inputs to GENSCAN. The results are documented in Table 4.15.

Table 4.15: Tests with GENSCAN to assess divergences between FP and FN

Description	Type	Length	Result
ENST00000441855.1	TN	514	No genes / exons found
ENST00000611726.1	TN	501	No genes / exons found
ENST00000431186.2	FP	522	1 exon found (0.514)
ENST00000529862.1	FP	507	1 exon found (0.962)

As the table suggests, GENSCAN identified the sequences as probable exons. This supports our theory that non-coding regions with similarity to coding regions most likely contributed to the misclassification divergences that we experienced in our tests. This didn't prove to be much of a nuisance fortunately, since pseudogenes and unprocessed pseudogenes were a minority in the original non-coding file.

4.4 Chapter Summary

In this chapter, we explained how we implemented our model by presenting its configuration and the reasoning behind it. We then reported the results of the tests that were made: one with the unprocessed dataset and another one with feature reduction after processing with PCA. We finished this section by assessing and discussing the results in the context of our problem.

Chapter 5

Conclusions

5.1 About the Developed Work

The main goal of this project was to assess how DL models could be applied in the classification of DNA sequences. Since gene prediction and gene annotation are essential tasks to understand how the various genomes work, we wanted to contribute to the area with some insights about trending technological approaches.

Developing a state of the art gene prediction software is not easy. Classifying genes, which was the main purpose behind our work, is just a small step towards a full developed tool. Every gene prediction software that has any usage in the bioinformatics field need to accomplish lots of features besides identifying gene sequences. This include, among others, the identification of promoter, terminator and regulatory regions, classification of homologous sequences and recognition of other specific binding sites within the genome. Usually there is also combined approaches between ab initio and comparative methods to enrich prediction results. To wrap all the technological capabilities, most gene prediction software also have an interface where the user can either change the existing parameters or see requested results. All of this proved to be impossible to achieve given our time limitations. However, with this work and research we wanted to assess and reinforce the usage of deep learning algorithms in the solving of genomic problems that still trouble science, and we think we were successful on that matter.

One problem that proved to be hard to overcome was all the computational restrictions that we had. The most popular DL projects usually deliver state of the art results by running large amounts of data (either in size and dimension) by long periods of time in clusters of dozens or even hundreds of high-end machines, all of them using GPUs for matrix calculations. By working with one machine through the use of its quad-core CPU, we had to drastically reduce the dimensionality of our project to make it solvable in due course.

5.2 Future Work

Regarding future prospects for this work, we could start by testing other existing architectures that have been receiving attention for their results. RNNs in particular have been popular in sequence classification problems with samples of varying length. They are however known to be particular hard to tune and usually take longer to converge comparing with FNNs such as our MLP. This facts alone led us to exclude this architecture still in the research phase of our work. Other considered option was a SDA, which would find alternative representations of our data through the introduction of noise by stacking autoencoders. Just as RNNs though, due to the nature of our data we would need a larger dataset to achieve state of the art results. This would result in a very slow training, not applicable to our time limitations.

More attempts at extracting other features from DNA sequences before feeding them to the models could also be made. This could result in overall more accurate evaluations.

The major improvement would have to be in our working environment. With achievements such as GPU usage for calculations or the integration of our machine in a cluster of other computers, we could improve our hyperparameters such as a reduced learning rate, train additional state of the art architectures or dramatically increase the size of our dataset. All of this could result in better outcomes than those we attained.

References

[AB93] Francisco Antequera and Adrian Bird. Number of cpg islands and genes in human and mouse. *Proceedings of the National Academy of Sciences*, 90(24):11995–11999, 1993.

[ABC+16] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. Tensorflow: a system for large-scale machine learning. In *OSDI*, volume 16, pages 265–283, 2016.

[AH12] Fareed Akthar and Caroline Hahne. Rapidminer 5 operator reference. *Rapid-I GmbH*, 2012.

[AJ3] Chainer: A flexible framework for neural networks. https://chainer.org/. Accessed: June 2018.

[AR02] Gautam Aggarwal and Ramakrishna Ramaswamy. Ab initio gene identification: prokaryote genome annotation with genescan and glimmer. *Journal of biosciences*, 27(1):7–14, 2002.

[AS+94] Rakesh Agrawal, Ramakrishnan Srikant, et al. Fast algorithms for mining association rules. In *Proc. 20th int. conf. very large data bases, VLDB*, volume 1215, pages 487–499, 1994.

[ATS03] Constantin F Aliferis, Ioannis Tsamardinos, and Alexander Statnikov. Hiton: a novel markov blanket algorithm for optimal variable selection. In *AMIA Annual Symposium Proceedings*, volume 2003, page 21. American Medical Informatics Association, 2003.

[AW10] Hervé Abdi and Lynne J Williams. Principal component analysis. *Wiley interdisciplinary reviews: computational statistics*, 2(4):433–459, 2010.

[AWJP08] Istvan Albert, Shinichiro Wachi, Cizhong Jiang, and B Franklin Pugh. Genetrack—a genomic data processing and visualization framework. *Bioinformatics*, 24(10):1305–1306, 2008.

[Bal12] Pierre Baldi. Autoencoders, unsupervised learning, and deep architectures. In *Proceedings of ICML workshop on unsupervised and transfer learning*, pages 37–49, 2012.

[bat] A gentle introduction to mini-batch gradient descent and how to configure batch size, jason brownlee. https://machinelearningmastery.com/gentle-introduction-mini-batch-gradient-descent-configure-batch-size/. Accessed: June 2018.

[BCD04] Ewan Birney, Michele Clamp, and Richard Durbin. Genewise and genomewise. *Genome research*, 14(5):988–995, 2004.

[BFAW+05] Carlos D Bustamante, Adi Fledel-Alon, Scott Williamson, Rasmus Nielsen, Melissa Todd Hubisz, Stephen Glanowski, David M Tanenbaum, Thomas J White, John J Sninsky, Ryan D Hernandez, et al. Natural selection on protein-coding genes in the human genome. *Nature*, 437(7062):1153, 2005.

[BG94] George Bebis and Michael Georgiopoulos. Feed-forward neural networks. *IEEE Potentials*, 13(4):27–31, 1994.

[BJCF07] Steven M Beitzel, Eric C Jensen, Abdur Chowdhury, and Ophir Frieder. Varying approaches to topical web query classification. In *Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 783–784. ACM, 2007.

[BKML+08] Dennis A Benson, Ilene Karsch-Mizrachi, David J Lipman, James Ostell, and David L Wheeler. Genbank. *Nucleic acids research*, 36(Database issue):D25, 2008.

[BYS+11] Dechao Bu, Kuntao Yu, Silong Sun, Chaoyong Xie, Geir Skogerbø, Ruoyu Miao, Hui Xiao, Qi Liao, Haitao Luo, Guoguang Zhao, et al. Noncode v3. 0: integrative annotation of long noncoding rnas. *Nucleic acids research*, 40(D1):D210–D215, 2011.

[CCK+00] Pete Chapman, Julian Clinton, Randy Kerber, Thomas Khabaza, Thomas Reinartz, Colin Shearer, and Rudiger Wirth. Crisp-dm 1.0 step-by-step data mining guide. 2000.

[Cel] What is a cell? - genetics home references. https://ghr.nlm.nih.gov/primer/basics/cell. Accessed: January 2018.

[CFG+09] Peter JA Cock, Christopher J Fields, Naohisa Goto, Michael L Heuer, and Peter M Rice. The sanger fastq file format for sequences with quality scores, and the solexa/illumina fastq variants. *Nucleic acids research*, 38(6):1767–1771, 2009.

[Cho17] François Chollet. Xception: Deep learning with depthwise separable convolutions. *arXiv preprint*, pages 1610–02357, 2017.

[CMP03] Francis S Collins, Michael Morgan, and Aristides Patrinos. The human genome project: lessons from large-scale biology. *Science*, 300(5617):286–290, 2003.

[Cri68] Francis HC Crick. The origin of the genetic code. *Journal of molecular biology*, 38(3):367–379, 1968.

[CSZ09] Olivier Chapelle, Bernhard Scholkopf, and Alexander Zien. Semi-supervised learning (chapelle, o. et al., eds.; 2006)[book reviews]. *IEEE Transactions on Neural Networks*, 20(3):542–542, 2009.

[CUH15] Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. Fast and accurate deep network learning by exponential linear units (elus). *arXiv preprint arXiv:1511.07289*, 2015.

[DBKMR05] Pieter-Tjerk De Boer, Dirk P Kroese, Shie Mannor, and Reuven Y Rubinstein. A tutorial on the cross-entropy method. *Annals of operations research*, 134(1):19–67, 2005.

[DC06] Jin Hwan Do and Dong-Kug Choi. Computational approaches to gene prediction. *The Journal of Microbiology*, 44(2):137–144, 2006.

[DG11] Sebastian Deorowicz and Szymon Grabowski. Compression of dna sequence reads in fastq format. *Bioinformatics*, 27(6):860–862, 2011.

[DJB+12] Thomas Derrien, Rory Johnson, Giovanni Bussotti, Andrea Tanzer, Sarah Djebali, Hagen Tilgner, Gregory Guernec, David Martin, Angelika Merkel, David G Knowles, et al. The gencode v7 catalog of human long noncoding rnas: analysis of their gene structure, evolution, and expression. *Genome research*, 22(9):1775–1789, 2012.

[DL4] Deeplearning4j: Open-source, distributed deep learning for the jvm. https://deeplearning4j.org/. Accessed: June 2018.

[Don11] Ciro Donalek. Supervised and unsupervised learning. In *Astronomy Colloquia. USA*, 2011.

[DSH13] George E Dahl, Tara N Sainath, and Geoffrey E Hinton. Improving deep neural networks for lvcsr using rectified linear units and dropout. In *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*, pages 8609–8613. IEEE, 2013.

[DVB93] Jacques De Villiers and Etienne Barnard. Backpropagation neural nets with one and two hidden layers. *IEEE transactions on neural networks*, 4(1):136–141, 1993.

[Eri15] Eric D. Green, James D. Watson and Francis S. Collins. Human Genome Project: Twenty-five years of big biology : Nature News. https://www.nature.com/news/human-genome-project-twenty-five-years-of-big-biology-1.18436, September 2015.

[For73] G David Forney. The viterbi algorithm. *Proceedings of the IEEE*, 61(3):268–278, 1973.

[FPSS96] Usama Fayyad, Gregory Piatetsky-Shapiro, and Padhraic Smyth. The kdd process for extracting useful knowledge from volumes of data. *Communications of the ACM*, 39(11):27–34, 1996.

[FRA] Comparison of frameworks | skymind. https://skymind.ai/wiki/comparison-frameworks-dl4j-tensorflow-pytorch. Accessed: June 2018.

[GB10] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256, 2010.

[GBB11a] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 315–323, 2011.

[GBB11b] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Domain adaptation for large-scale sentiment classification: A deep learning approach. In *Proceedings of the 28th international conference on machine learning (ICML-11)*, pages 513–520, 2011.

[GCW+96] Robert Gaizauskas, Hamish Cunningham, Yorick Wilks, Peter Rodgers, and Kevin Humphreys. Gate: An environment to supportresearch and development in natural language engineering. In *ictai*, page 58. IEEE, 1996.

[GD98] Matt W Gardner and SR Dorling. Artificial neural networks (the multilayer perceptron)—a review of applications in the atmospheric sciences. *Atmospheric environment*, 32(14-15):2627–2636, 1998.

[GEN] Ncbi-genbank flat file release 226.0. ftp://ftp.ncbi.nih.gov/genbank/gbrel.txt. Accessed: June 2018.

[GMH13] Alex Graves, Abdel-rahman Mohamed, and Geoffrey Hinton. Speech recognition with deep recurrent neural networks. In *Acoustics, speech and signal processing (icassp), 2013 ieee international conference on*, pages 6645–6649. IEEE, 2013.

[Gro13] Stephen Grossberg. Recurrent neural networks. *Scholarpedia*, 8(2):1888, 2013.

[Ham94] James Douglas Hamilton. *Time series analysis*, volume 2. Princeton university press Princeton, NJ, 1994.

[Han07] David J Hand. Principles of data mining. *Drug safety*, 30(7):621–622, 2007.

[Haw04] Douglas M Hawkins. The problem of overfitting. *Journal of chemical information and computer sciences*, 44(1):1–12, 2004.

[HBB+02] Tim Hubbard, Daniel Barker, Ewan Birney, Graham Cameron, Yuan Chen, L Clark, Tony Cox, J Cuff, Val Curwen, Thomas Down, et al. The ensembl genome database project. *Nucleic acids research*, 30(1):38–41, 2002.

[HCBS03] Alexander K Hudek, Joseph Cheung, Andrew P Boright, and Stephen W Scherer. Genescript: Dna sequence annotation pipeline. *Bioinformatics*, 19(9):1177–1178, 2003.

[HFH+09] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H Witten. The weka data mining software: an update. *ACM SIGKDD explorations newsletter*, 11(1):10–18, 2009.

[Hin92] Geoffrey E Hinton. How neural networks learn from experience. *Scientific American*, 267(3):144–151, 1992.

[HPK11] Jiawei Han, Jian Pei, and Micheline Kamber. *Data mining: concepts and techniques*. Elsevier, 2011.

[IJD] Torch | scientific computing for luajit. http://torch.ch/. Accessed: June 2018.

[JSD+14] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional architecture for fast feature embedding. In *Proceedings of the 22nd ACM international conference on Multimedia*, pages 675–678. ACM, 2014.

[KAH+14] Sasan Karamizadeh, Shahidan M Abdullah, Mehran Halimi, Jafar Shayan, and Mohammad javad Rajabi. Advantage and drawback of support vector machine functionality. In *Computer, Communications, and Control Technology (I4CT), 2014 International Conference on*, pages 63–65. IEEE, 2014.

[KB14] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[KFDB01] Ian Korf, Paul Flicek, Daniel Duan, and Michael R Brent. Integrating genomic homology into gene structure prediction. *Bioinformatics*, 17(suppl_1):S140–S148, 2001.

[Kim14] Yoon Kim. Convolutional neural networks for sentence classification. *arXiv preprint arXiv:1408.5882*, 2014.

[KK92] Barry L Kalman and Stan C Kwasny. Why tanh: choosing a sigmoidal function. In *Neural Networks, 1992. IJCNN., International Joint Conference on*, volume 4, pages 578–581. IEEE, 1992.

[KLSS17] Nataliia Kussul, Mykola Lavreniuk, Sergii Skakun, and Andrii Shelestov. Deep learning classification of land cover and crop types using remote sensing data. *IEEE Geoscience and Remote Sensing Letters*, 14(5):778–782, 2017.

[Kor04] Ian Korf. Gene finding in novel genomes. *BMC bioinformatics*, 5(1):59, 2004.

[Koz83] Marilyn Kozak. Comparison of initiation of protein synthesis in procaryotes, eucaryotes, and organelles. *Microbiological reviews*, 47(1):1, 1983.

[KSH12] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[KVJK05] Onkar C. Kulkarni, R. Vigneshwar, Valadi K. Jayaraman, and Bhaskar D. Kulkarni. Identification of coding and non-coding sequences using local hölder exponent formalism. *Bioinformatics*, 21(20):3818–3823, 2005.

[L+15] Yann LeCun et al. Lenet-5, convolutional neural networks. *URL: http://yann. lecun. com/exdb/lenet*, page 20, 2015.

[Lan] R vs. java vs. python: which is right for your project? https://www.upwork.com/hiring/data/r-vs-java-vs-python-which-is-best/. Accessed: June 2018.

[LBZ+95] Harvey Lodish, Arnold Berk, S Lawrence Zipursky, Paul Matsudaira, David Baltimore, James Darnell, et al. *Molecular cell biology*, volume 3. WH Freeman New York, 1995.

[LDR00] Jinyan Li, Guozhu Dong, and Kotagiri Ramamohanarao. Instance-based classification by emerging patterns. In *European Conference on Principles of Data Mining and Knowledge Discovery*, pages 191–200. Springer, 2000.

[Lej01] Miguel APM Lejeune. Measuring the impact of data mining on churn management. *Internet Research*, 11(5):375–387, 2001.

[Len02] Maurizio Lenzerini. Data integration: A theoretical perspective. In *Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 233–246. ACM, 2002.

[LNC+11] Quoc V Le, Jiquan Ngiam, Adam Coates, Abhik Lahiri, Bobby Prochnow, and Andrew Y Ng. On optimization methods for deep learning. In *Proceedings of the 28th International Conference on International Conference on Machine Learning*, pages 265–272. Omnipress, 2011.

[Lop12] R Lopez. Opennn: Open neural networks library (version 0.9), 2012.

[LRB08] Nicolas Le Roux and Yoshua Bengio. Representational power of restricted boltzmann machines and deep belief networks. *Neural computation*, 20(6):1631–1649, 2008.

[LS17] Paras Lakhani and Baskaran Sundaram. Deep learning at chest radiography: automated classification of pulmonary tuberculosis by using convolutional neural networks. *Radiology*, 284(2):574–582, 2017.

[M+06] Kevin P Murphy et al. Naive bayes classifiers. *University of British Columbia*, 18, 2006.

[Mar10] James Martens. Deep learning via hessian-free optimization. In *ICML*, volume 27, pages 735–742, 2010.

[MD02] Irmtraud M Meyer and Richard Durbin. Comparative ab initio prediction of gene structures using pair hmms. *Bioinformatics*, 18(10):1309–1318, 2002.

[MDKF08] André C Marreiros, Jean Daunizeau, Stefan J Kiebel, and Karl J Friston. Population dynamics: variance and the sigmoid activation function. *Neuroimage*, 42(1):147–157, 2008.

[MIK] Plaidml. https://github.com/plaidml. Accessed: June 2018.

[MMG12] Sharmila S Mande, Monzoorul Haque Mohammed, and Tarini Shankar Ghosh. Classification of metagenomic sequences: methods and challenges. *Briefings in bioinformatics*, 13(6):669–681, 2012.

[MSSR02] Catherine Mathé, Marie-France Sagot, Thomas Schiex, and Pierre Rouzé. Current methods of gene prediction, their strengths and weaknesses. *Nucleic acids research*, 30(19):4103–4117, 2002.

[NH10] Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 807–814, 2010.

[NK05] Pernille Nielsen and Anders Krogh. Large-scale prokaryotic gene prediction and comparison to genome annotation. *Bioinformatics*, 21(24):4322–4329, 2005.

[NTN+16] Ngoc Giang Nguyen, Vu Anh Tran, Duc Luu Ngo, Dau Phan, Favorisen Rosyking Lumbanraja, Mohammad Reza Faisal, Bahriddin Abapihi, Mamoru Kubo, and Kenji Satou. Dna sequence classification by convolutional neural network. *Journal of Biomedical Science and Engineering*, 9(05):280, 2016.

[OIJ] Mxnet: A scalable deep learning framework. https://mxnet.apache.org/. Accessed: June 2018.

[Out] How does deep learning outperform other machine learning algorithms? https://medium.com/@haohanwang/how-deep-learning-outperforms-other-machine-learning-algorithms-fdfd4e55fcf3. Accessed: June 2018.

[PBH+94] C-K Peng, Sergey V Buldyrev, Shlomo Havlin, Michael Simons, H Eugene Stanley, and Ary L Goldberger. Mosaic organization of dna nucleotides. *Physical review e*, 49(2):1685, 1994.

[PC16] Maximilian Panzner and Philipp Cimiano. Comparing hidden markov models and long short term memory neural networks for learning action representations. In *International Workshop on Machine Learning, Optimization and Big Data*, pages 94–105. Springer, 2016.

[PIM+10] Amrita Pati, Natalia N Ivanova, Natalia Mikhailova, Galina Ovchinnikova, Sean D Hooper, Athanasios Lykidis, and Nikos C Kyrpides. Geneprimp: a gene prediction improvement pipeline for prokaryotic genomes. *Nature methods*, 7(6):455, 2010.

[Pla] Classifying plankton with deep neural networks, sander dieleman. http://benanne.github.io/2015/03/17/plankton.html. Accessed: June 2018.

[PM92] Sankar K Pal and Sushmita Mitra. Multilayer perceptron, fuzzy sets, classifiaction. 1992.

[PRBB17] Daewoo Pak, Robert Root-Bernstein, and Zachary F Burton. trna structure and evolution and standardization to the three nucleotide genetic code. *Transcription*, 8(4):205–219, 2017.

[RKA06] Juan José Rodriguez, Ludmila I Kuncheva, and Carlos J Alonso. Rotation forest: A new classifier ensemble method. *IEEE transactions on pattern analysis and machine intelligence*, 28(10):1619–1630, 2006.

[RM08] Lior Rokach and Oded Z Maimon. *Data mining with decision trees: theory and applications*, volume 69. World scientific, 2008.

[RPR] R: What is r? https://www.r-project.org/about.html. Accessed: June 2018.

[SAM] Sample genbank record. https://www.ncbi.nlm.nih.gov/Sitemap/samplerecord.html. Accessed: June 2018.

[Sch15] Jürgen Schmidhuber. Deep learning in neural networks: An overview. *Neural networks*, 61:85–117, 2015.

[SEH16] Virag Sharma, Anas Elghafari, and Michael Hiller. Coding exon-structure aware realigner (cesar) utilizes genome alignments for accurate comparative gene annotation. *Nucleic acids research*, 44(11):e103–e103, 2016.

[SFWS18] Bonan Song, Chunxiao Fan, Yuexin Wu, and Juanjuan Sun. Data prediction for public events in professional domains based on improved rnn-lstm. In *Journal of Physics: Conference Series*, volume 976, page 012007. IOP Publishing, 2018.

[SHK+14] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[Sle10] Roy D Sleator. An overview of the current status of eukaryote gene prediction strategies. *Gene*, 461(1):1–4, 2010.

[SM05] Mario Stanke and Burkhard Morgenstern. Augustus: a web server for gene prediction in eukaryotes that allows user-defined constraints. *Nucleic acids research*, 33(suppl_2):W465–W467, 2005.

[SN00] David Sankoff and Joseph H Nadeau. Comparative genomics. In *Comparative Genomics*, pages 3–7. Springer, 2000.

[Sta09] D Stathakis. How many hidden layers and nodes? *International Journal of Remote Sensing*, 30(8):2133–2147, 2009.

[SWT+98] Larry Simpson, Shirley H Wang, Otavio H Thiemann, Juan D Alfonzo, Dmitri A Maslov, and Herbert A Avila. U-insertion/deletion edited sequence database. *Nucleic acids research*, 26(1):170–176, 1998.

[T39] Keras documentation. https://keras.io/. Accessed: June 2018.

[Tan13] Yichuan Tang. Deep learning using linear support vector machines. *arXiv preprint arXiv:1306.0239*, 2013.

[TRCM08] Herve Tettelin, David Riley, Ciro Cattuto, and Duccio Medini. Comparative genomics: the bacterial pan-genome. *Current opinion in microbiology*, 11(5):472–477, 2008.

[VAM+01] J Craig Venter, Mark D Adams, Eugene W Myers, Peter W Li, Richard J Mural, Granger G Sutton, Hamilton O Smith, Mark Yandell, Cheryl A Evans, Robert A Holt, et al. The sequence of the human genome. *science*, 291(5507):1304–1351, 2001.

[VBY+95] Pieter Vermeulen, Etienne Barnard, Yonghong Yan, Mark Fanty, and Ronald Coley. A comparison of hmm and neural network approaches to real world telephone speech applications. In *IEEE International Conference on Neural Networks and Signal Processing*, pages 796–799. Citeseer, 1995.

[Vie13] Propieties Viewer. Chemicalize. org. *Recuperado el*, 5, 2013.

[VVDVDV+02] Laura J Van't Veer, Hongyue Dai, Marc J Van De Vijver, Yudong D He, Augustinus AM Hart, Mao Mao, Hans L Peterse, Karin Van Der Kooy, Matthew J Marton, Anke T Witteveen, et al. Gene expression profiling predicts clinical outcome of breast cancer. *nature*, 415(6871):530, 2002.

[WB09] Zhou Wang and Alan C Bovik. Mean squared error: Love it or leave it? a new look at signal fidelity measures. *IEEE signal processing magazine*, 26(1):98–117, 2009.

[WZ04] Li Y Wang Z, Chen Y. A brief review of computational gene prediction methods. *Genomics, Proteomics and Bioinformatics*, pages 216–221, November 2004.

[XKS+06] Xiaopeng Xi, Eamonn Keogh, Christian Shelton, Li Wei, and Chotirat Ann Ratanamahatana. Fast time series classification using numerosity reduction. In *Proceedings of the 23rd international conference on Machine learning*, pages 1033–1040. ACM, 2006.

[XWCL15] Bing Xu, Naiyan Wang, Tianqi Chen, and Mu Li. Empirical evaluation of rectified activations in convolutional network. *arXiv preprint arXiv:1505.00853*, 2015.

[Yeg09] B Yegnanarayana. *Artificial neural networks*. PHI Learning Pvt. Ltd., 2009.

[YKF+09] Moran Yassour, Tommy Kaplan, Hunter B Fraser, Joshua Z Levin, Jenna Pfiffner, Xian Adiconis, Gary Schroth, Shujun Luo, Irina Khrebtukova, Andreas Gnirke, et al. Ab initio construction of a eukaryotic transcriptome by massively parallel mrna sequencing. *Proceedings of the National Academy of Sciences*, 106(9):3264–3269, 2009.

[Yun76] Jorge J Yunis. High resolution of human chromosomes. *Science*, pages 1268–1270, 1976.

[ZUA] Microsoft cognitive toolkit. https://www.microsoft.com/en-us/cognitive-toolkit/. Accessed: June 2018.

[ZZ05] Min-Ling Zhang and Zhi-Hua Zhou. A k-nearest neighbor based algorithm for multi-label classification. In *Granular Computing, 2005 IEEE International Conference on*, volume 2, pages 718–721. IEEE, 2005.